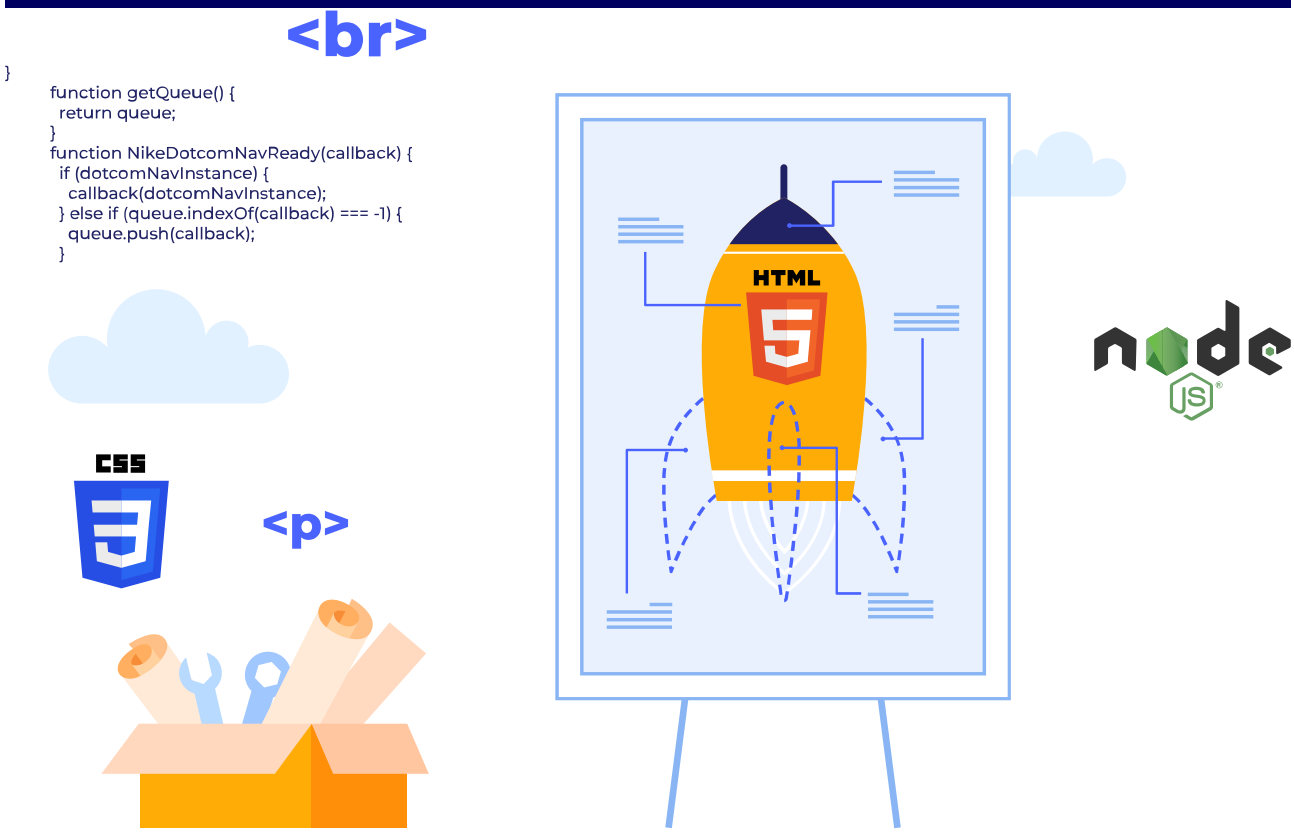
Programación Web



José Luis Quiroz Fabián

Índice

1. JavaScript **4**

1.1. ¿Cómo se declara código JavaScript? . . . 4
1.2. Desplegar en consola y mensaje de alerta . . . 4
1.3. Tipos de datos y variables . . . 5
1.4. Funciones . . . 7
1.4.1. Objeto arguments . . . 7
1.4.2. Parámetro por defecto . . . 8
1.4.3. Ejemplo de funciones . . . 8
1.4.4. Recursividad . . . 11
1.4.5. Ejercicio . . . 12
1.5. Estructuras de control . . . 14
1.5.1. Estructuras de condición . . . 14
1.5.2. Estructuras de repetición . . . 17
1.6. Ejercicio . . . 19
1.7. Clases . . . 19
1.7.1. Definición de una clase con **class** . . . 19
1.7.2. Definición de una clase con **function** . . . 21
1.7.3. Herencia . . . 23
1.7.4. Sobrecarga . . . 25
1.8. Document Object Model (DOM) . . . 29
1.8.1. Encontrando elemento por id . . . 30
1.8.2. Encontrando elementos por nombre de etiqueta . . . 31
1.8.3. Encontrando elementos por nombre de clase . . . 33
1.8.4. Encontrando elementos selector CSS . . . 34
1.8.5. Encontrando elementos de una colección form . . . 35
1.8.6. Cambiando el contenido de un elemento HTML . . . 36
1.8.7. Cambiando el valor de un atributo . . . 37
1.8.8. Cambiando un estilo . . . 38
1.8.9. Ejercicio: Reloj . . . 41
1.9. Eventos del teclado y del ratón . . . 42
1.10. APIs HTML5 . . . 45
1.11. Canvas . . . 46
1.11.1. El área del canvas . . . 46
1.11.2. Una línea en canvas . . . 47
1.11.3. Un círculo en canvas . . . 48
1.11.4. Una animación . . . 49

1.12. Drag and drop 49
1.12.1. Eventos Drag&Drop 49
1.12.2. Eventos 50
1.12.3. Arrastrar y soltar 51
1.12.4. Varios elementos 53
1.12.5. Ejercicio 57
1.13. Web Storage 57
1.13.1. Session Storage 57
1.13.2. Local Storage 60
1.13.3. Ejercicio 64

1 JavaScript

JavaScript es un lenguaje de alto nivel, dinámico, débilmente tipado, orientado a objetos (contiene una biblioteca estándar de objetos como por ejemplo: Array, Date, y Math), multiparadigma e interpretado. Junto con HTML y CSS, JavaScript es una de las tres tecnologías núcleo del contenido World Wide Web. Se utiliza para crear páginas interactivas y proporcionar programas online. Actualmente la mayoría de los sitios lo incluyen, y todos los navegadores modernos lo soportan sin necesidad de incluir un plug-ins lo que significa que mantienen un motor de ejecución JavaScript.

1.1 ¿Cómo se declara código JavaScript?

Puede ser declarando el código directamente:

```
<!DOCTYPE html>
<html>
   <body>
      <p></p>
      <script>
      </script>
   </body>
</html>
```

o mediante referencia a un archivo JavaScript que se declara comúnmente en la etiqueta **head**.

```
 <script src="myscripts.js"></script>
```

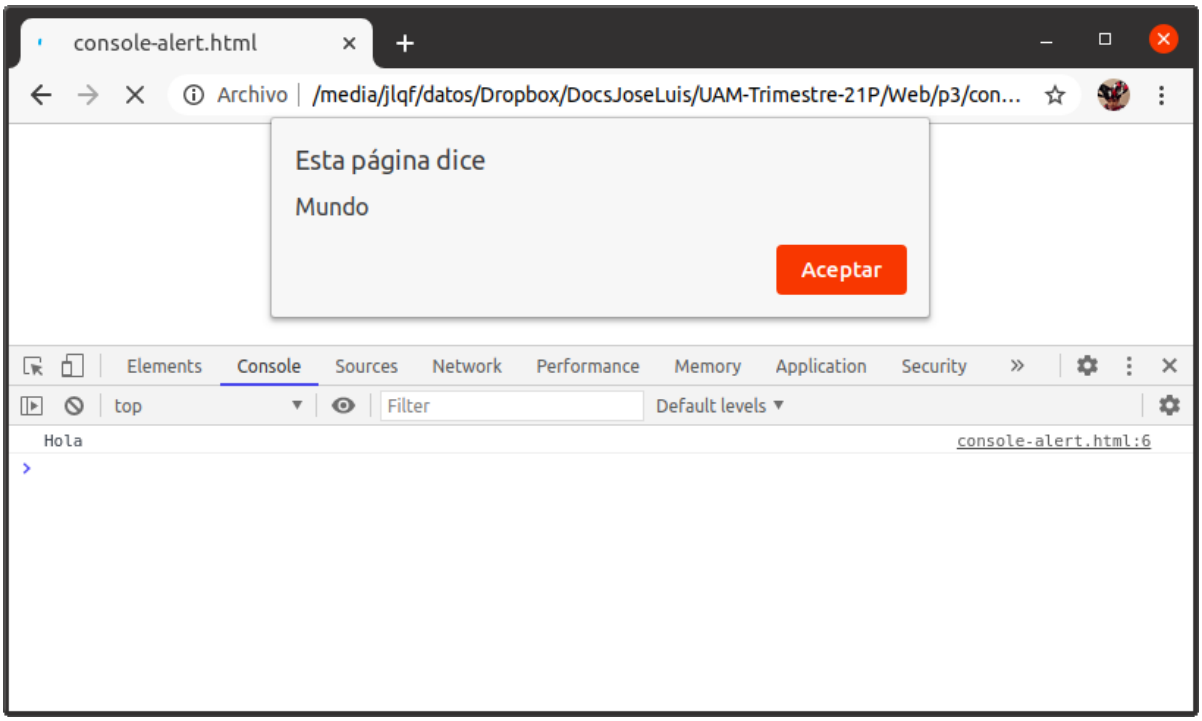
1.2 Desplegar en consola y mensaje de alerta

En JavaScript es muy útil desplegar información en la consola del navegador o mediante una ventana de alerta a fin de depurar nuestro código. Lo anterior se puede realizar de la siguiete forma:

```
<!DOCTYPE html>
<html>
   <body>
      <p></p>
      <script>
         console.log("Hola");
         alert("Mundo")
      </script>
   </body>
```

```
10 </html>
```

Salida:



1.3 Tipos de datos y variables

JavaScript contempla los siguientes tipos de datos primitivos

1. string.
2. number.
3. boolean.

Los tipos de datos compuestos (de referencia) son:

1. Object
2. Array

Para declarar una variable se utilizan las palabras reservadas:

- **var**: Variables de alcance global.
- **let**: Variables de alcance local.
- **const**: Variable que no puede ser reasignada (constante).

Las variables de JavaScript no tienen ningún tipo predeterminado (como int, double, float, etc), en cambio, el tipo de una variable es el tipo de su valor. Por ejemplo, en el Código 1 la variable *b* tiene originalmente asignado un valor de tipo booleano (línea 8) pero en la línea 13 se le asigna un valor de tipo numérico. La primera mención de la variable la define en la memoria, para que se pueda hacer referencia a ella más adelante en el script.

En JavaScript, se puede realizar operaciones con valores de tipos diferentes sin provocar una excepción. El intérprete de JavaScript convierte implícitamente uno de los tipos de datos en el otro tipo y, después, realiza la operación. Las reglas de conversión de los valores alfanuméricos, numéricos y booleanos son las siguientes:

1. Si se suman un número y una cadena, el número se convierte en una cadena.
2. Si se suman un valor booleano y una cadena, el tipo booleano se convierte en una cadena.
3. Si se suman un número y un valor booleano, el tipo booleano se convierte en un número.

Las variables pueden tomar valores especiales en su inicialización o declaración, los cuales son:

1. null
2. undefined

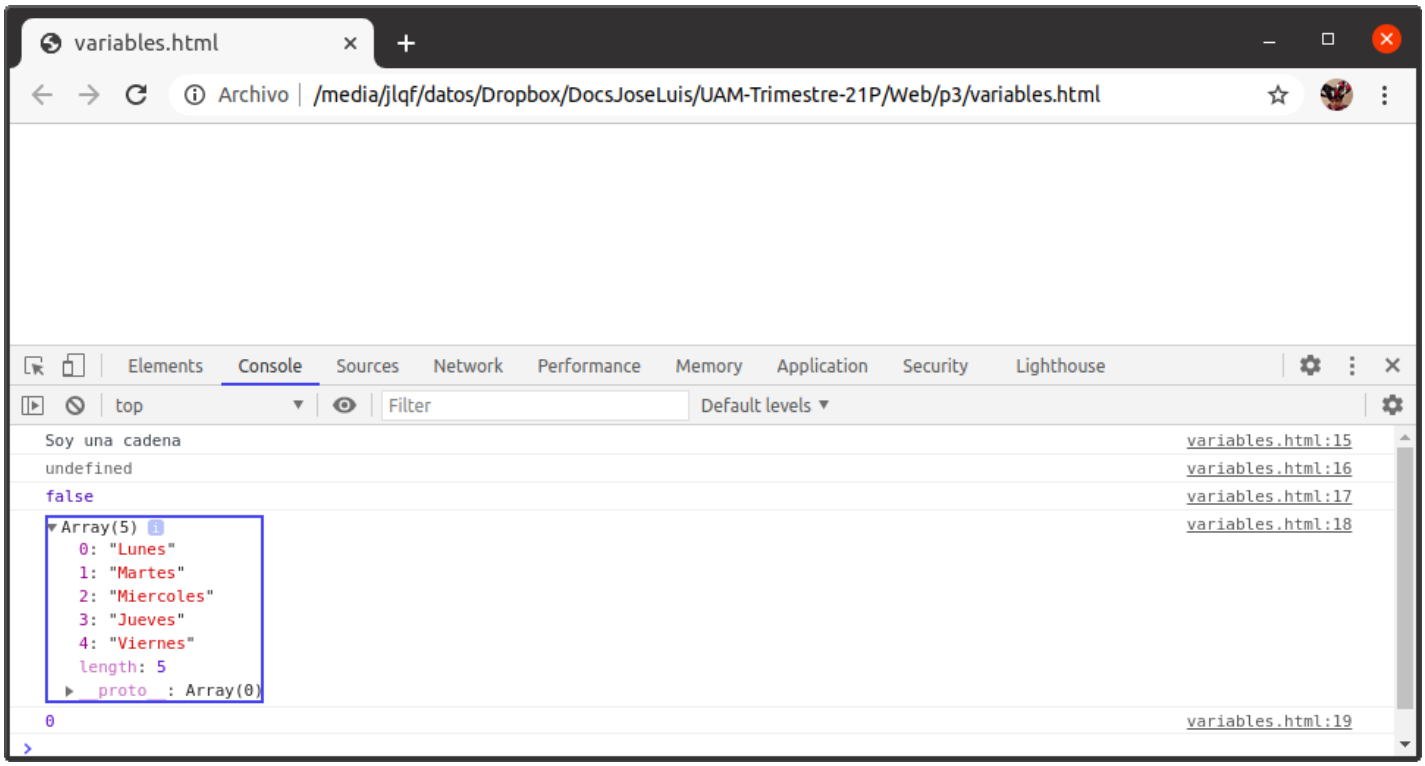
El primero es para inicializar una variable y el segundo es cuando a una variable no se le asigna un valor inicial.

```
1  <!DOCTYPE html>
2  <html>
3     <body>
4        <p></p>
5        <script>
6           var cad="Soy una cadena";
7           var x;
8           var b=false;
9
10          //Comentario: Arreglo de cadenas
11          dias=["Lunes","Martes","Miercoles","Jueves","Viernes"];
12
13          b=0
14
15          console.log(cad);
16          console.log(x);
17          console.log(b);
18          console.log(dias);
19          console.log(p);
20       </script>
```

```
21    </body>
22 </html>
```

Código 1: Tipos de datos

Salida:



1.4 Funciones

Las funciones en JavaScript se definen mediante la palabra reservada **function**, seguida del nombre de la función. Por ejemplo:

```

 function suma(a,b){
     c= a+b;
     return c;
}
```

Código 2: Definición de una función

1.4.1 Objeto arguments

Los argumentos de una función son mantenidos en un objeto similar a un array llamado **arguments**. El primer argumento pasado a una función sería **arguments[0]**. El número total de argumentos es mostrado por **arguments.length**.

1.4.2 Parámetro por defecto

En JS los parámetros de una función tienen por defecto un valor de **undefined**, pero se puede establecer un valor por defecto diferente de tal forma que si al invocar a la función no se mandan los parámetros completos estos asumen dicho valor.

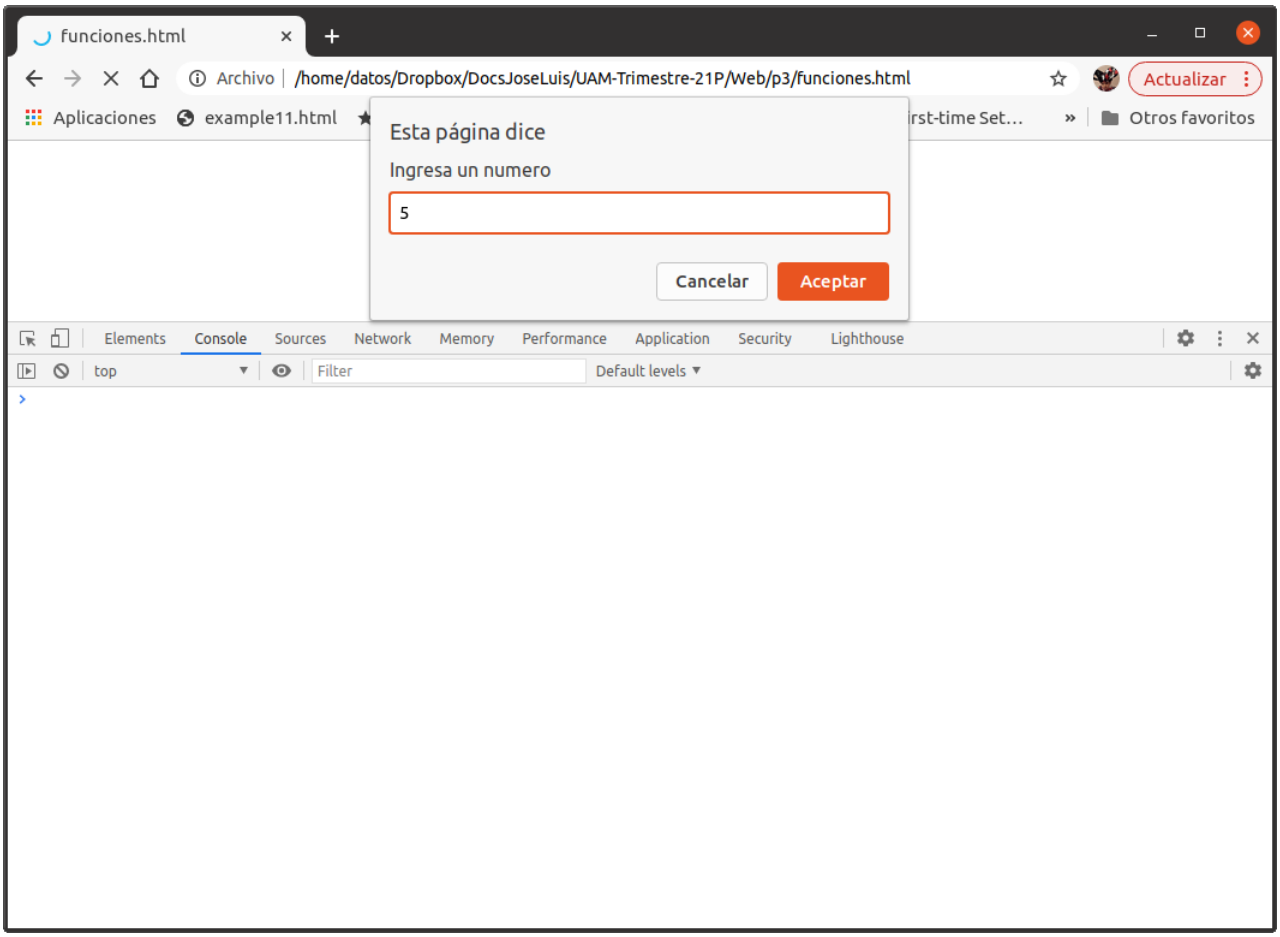
1.4.3 Ejemplo de funciones

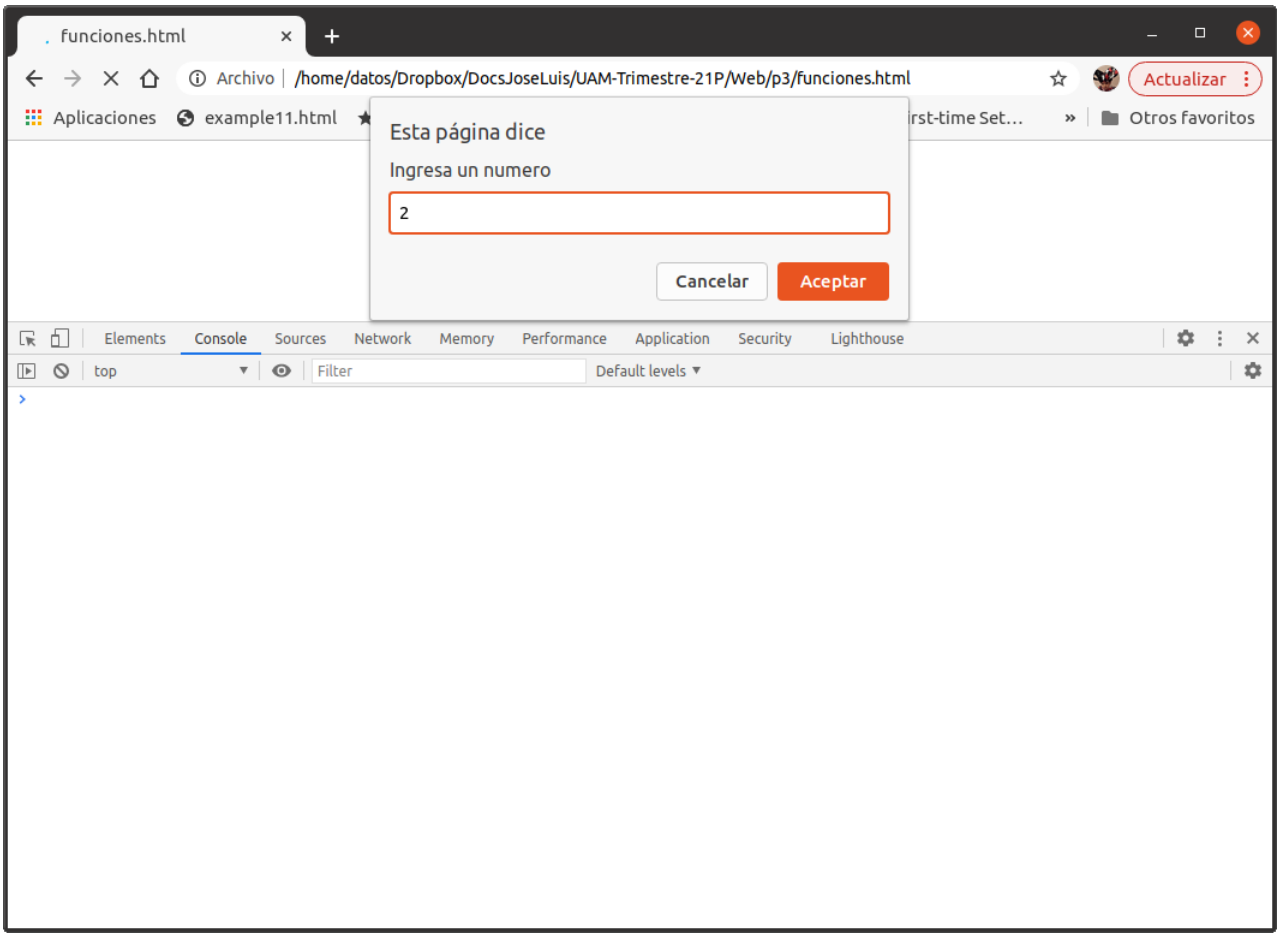
```
<!DOCTYPE html>
<html>
   <body>
      <p></p>
      <script>

         op1 = prompt("Ingresa un numero");
         op2 = prompt("Ingresa un numero");

         console.log("Operadores: "+op1+" , "+op2);

         intOp1=parseInt(op1);
         intOp2=parseInt(op2);

         r=resta(intOp1,intOp2);

         console.log("Resultado resta:"+r);

         suma(intOp1,intOp2);

         suma("4","6");

         suma(3,4,5);

         multiplicacion();

         multiplicacion(3,4);
         function resta(a,b){
            return a-b;
         }

         function suma(){

            console.log("#Argumentos:"+arguments.length)
            console.log("Argumento 1: "+arguments[0]);
            console.log("Argumento 2: "+arguments[1]);
            console.log("Argumento 3: "+arguments[2]);

```

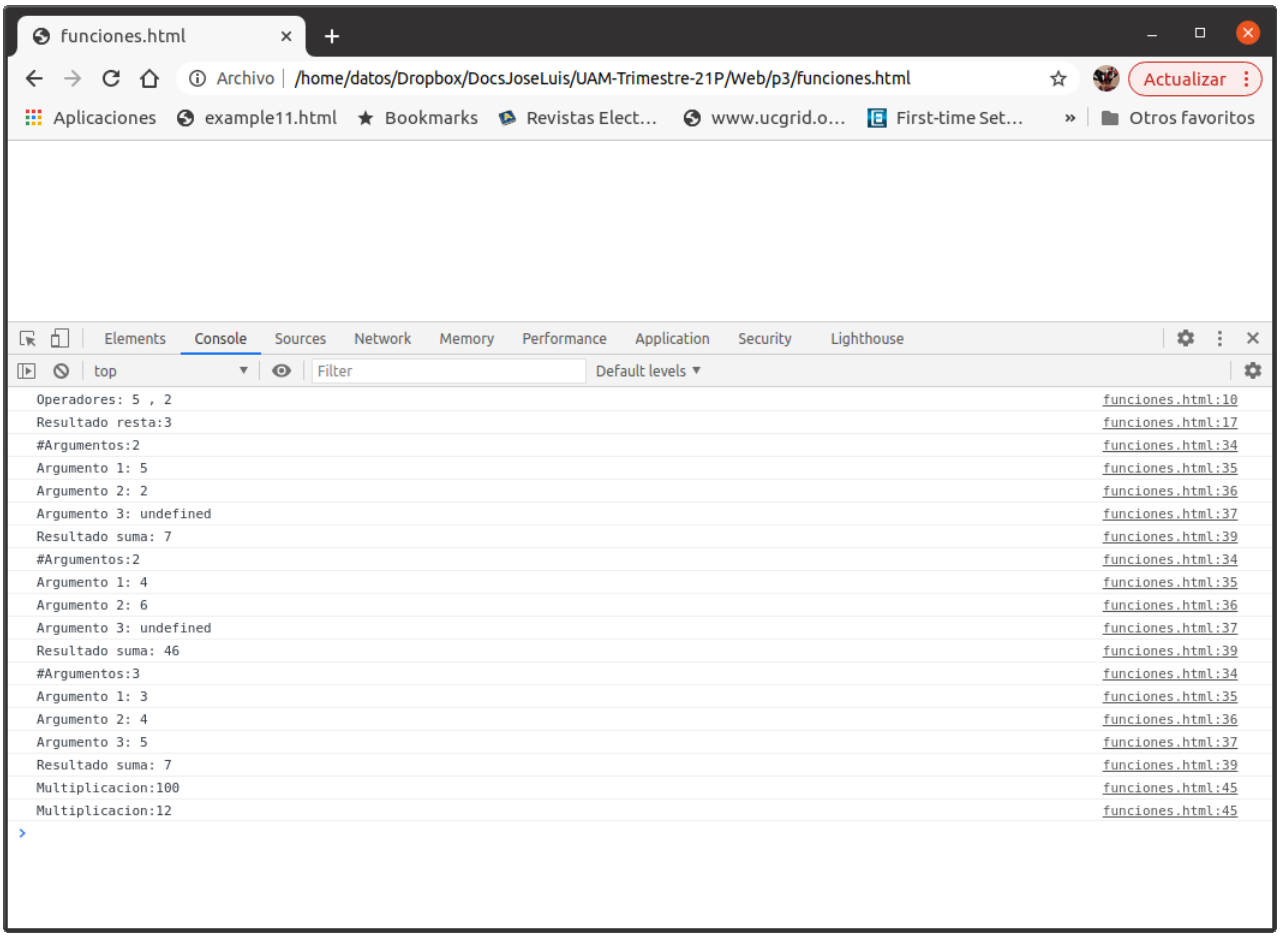
```
39              console.log("Resultado suma: "+(arguments[0]+arguments[1]));
40
41          }
42
43          function multiplicacion(a=5,b=20){
44
45              console.log("Multiplicacion:"+a*b);
46
47          }
48
49      </script>
50    </body>
51  </html>
```

Código 3: Ejemplo de funciones

Salida:




funciones.html
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p3/funciones.html
Actualizar
Aplicaciones
example11.html
Otros favoritos
Esta página dice
Ingresa un numero
2
Cancelar
Aceptar
Elements
Console
Sources
Network
Memory
Performance
Application
Security
Lighthouse
top
Filter
Default levels

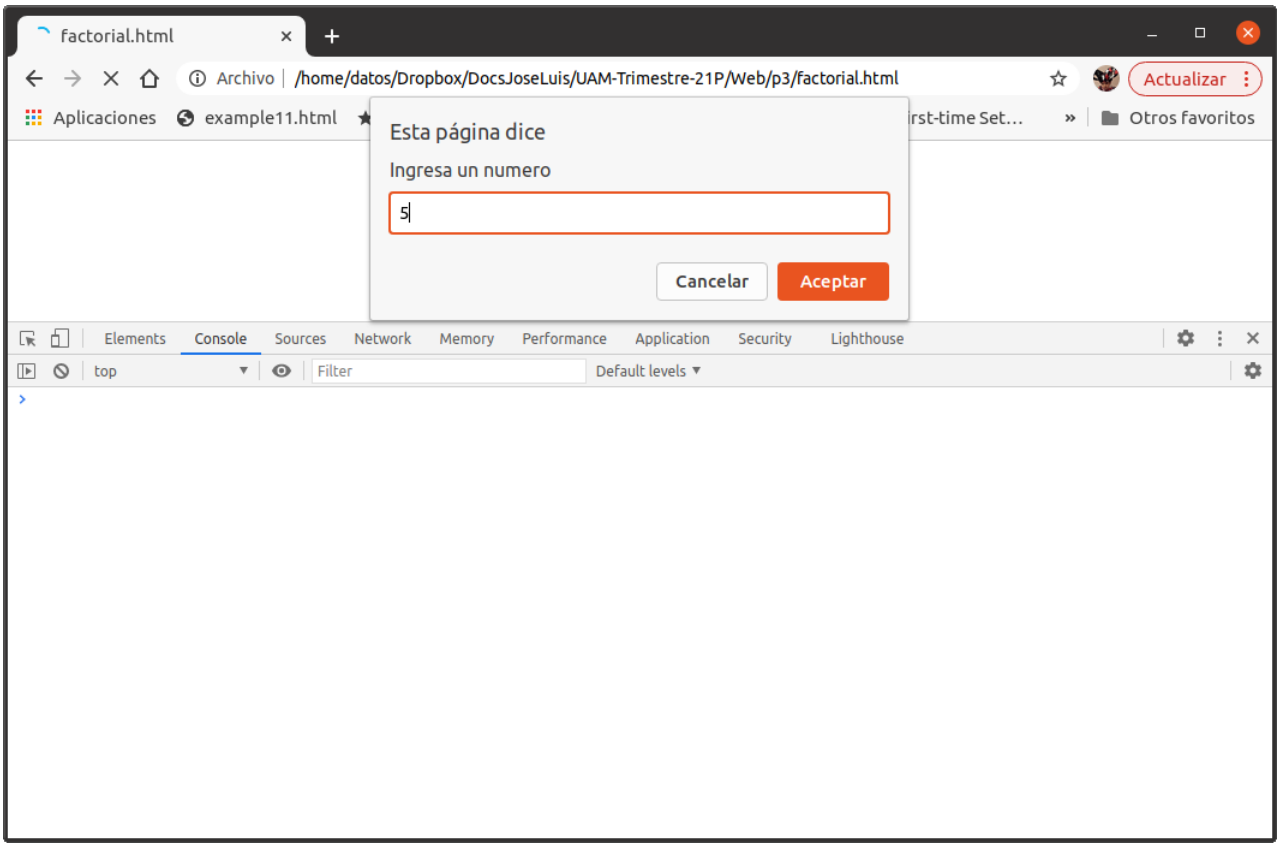

funciones.html
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p3/funciones.html
Actualizar
Aplicaciones
example11.html
Bookmarks
Revistas Elect...
www.ucgrid.o...
First-time Set...
Otros favoritos
Elements
Console
Sources
Network
Memory
Performance
Application
Security
Lighthouse
top
Filter
Default levels
Operadores: 5 , 2 funciones.html:10
Resultado resta:3 funciones.html:17
#Argumentos:2 funciones.html:34
Argumento 1: 5 funciones.html:35
Argumento 2: 2 funciones.html:36
Argumento 3: undefined funciones.html:37
Resultado suma: 7 funciones.html:39
#Argumentos:2 funciones.html:34
Argumento 1: 4 funciones.html:35
Argumento 2: 6 funciones.html:36
Argumento 3: undefined funciones.html:37
Resultado suma: 46 funciones.html:39
#Argumentos:3 funciones.html:34
Argumento 1: 3 funciones.html:35
Argumento 2: 4 funciones.html:36
Argumento 3: 5 funciones.html:37
Resultado suma: 7 funciones.html:39
Multiplicacion:100 funciones.html:45
Multiplicacion:12 funciones.html:45

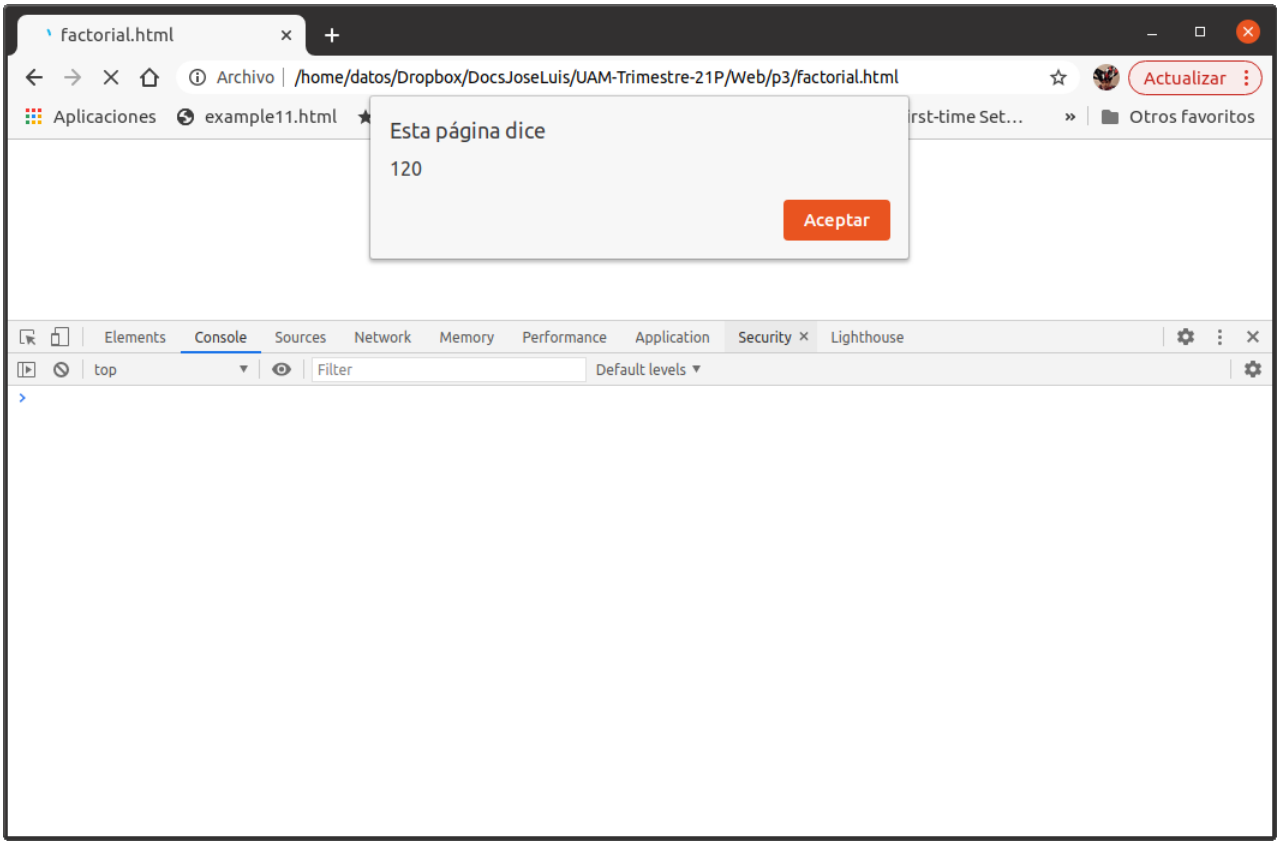
1.4.4 Recursividad

```
<!DOCTYPE html>
<html>
   <body>
      <p></p>
      <script>

         n = prompt("Ingresa un numero");

         r=factorial(n);

         alert(r);

         function factorial(n){
              (n==0 || n==1) && (fact=1);

              (n>1) && (fact=n*factorial(n-1));

              return fact;

         }

      </script>
   </body>
</html>
```

Código 4: Ejemplo de funciones

Salida:





1.4.5 Ejercicio

Mediante funciones y sin estructuras condicionales o de repetición calcular la aproximación:

$$e^x \approx \sum_{i=0}^{n} \frac{(x)^i}{(i!)}$$

Debe ingresar *n* y *x* mediante **prompt**.

```
<!DOCTYPE html>
<html>
   <body>
      <p></p>
      <script>

         n = prompt("Ingresa n");

         x = prompt("Ingresa x");

         r=aproximacion(parseInt(x),parseInt(n));

         alert(r);

         function factorial(n){
              (n==0 || n==1) && (fact=1);

              (n>1) && (fact=n*factorial(n-1));

              return fact;

         }

         function potencia(base,exp){
              (exp==0) && (pot=1);

              (exp==1) && (pot=base);

              (exp>1) && (pot=base*potencia(base,exp-1));

              return pot;

         }

          function aproximacion(x,n){
              (n==0) && (aprox=1);

              (n>0) && (aprox=potencia(x,n)/factorial(n)+aproximacion(x,n-1));

              return aprox;

         }
```

```
43
44
45          </script>
46      </body>
47  </html>
```

Código 5: Solución del ejericicio

1.5 Estructuras de control

1.5.1 Estructuras de condición

JavaScript soporta las estructuras **if**, **if-else** y **switch** con la sintaxis similar al del lenguaje C y Java.

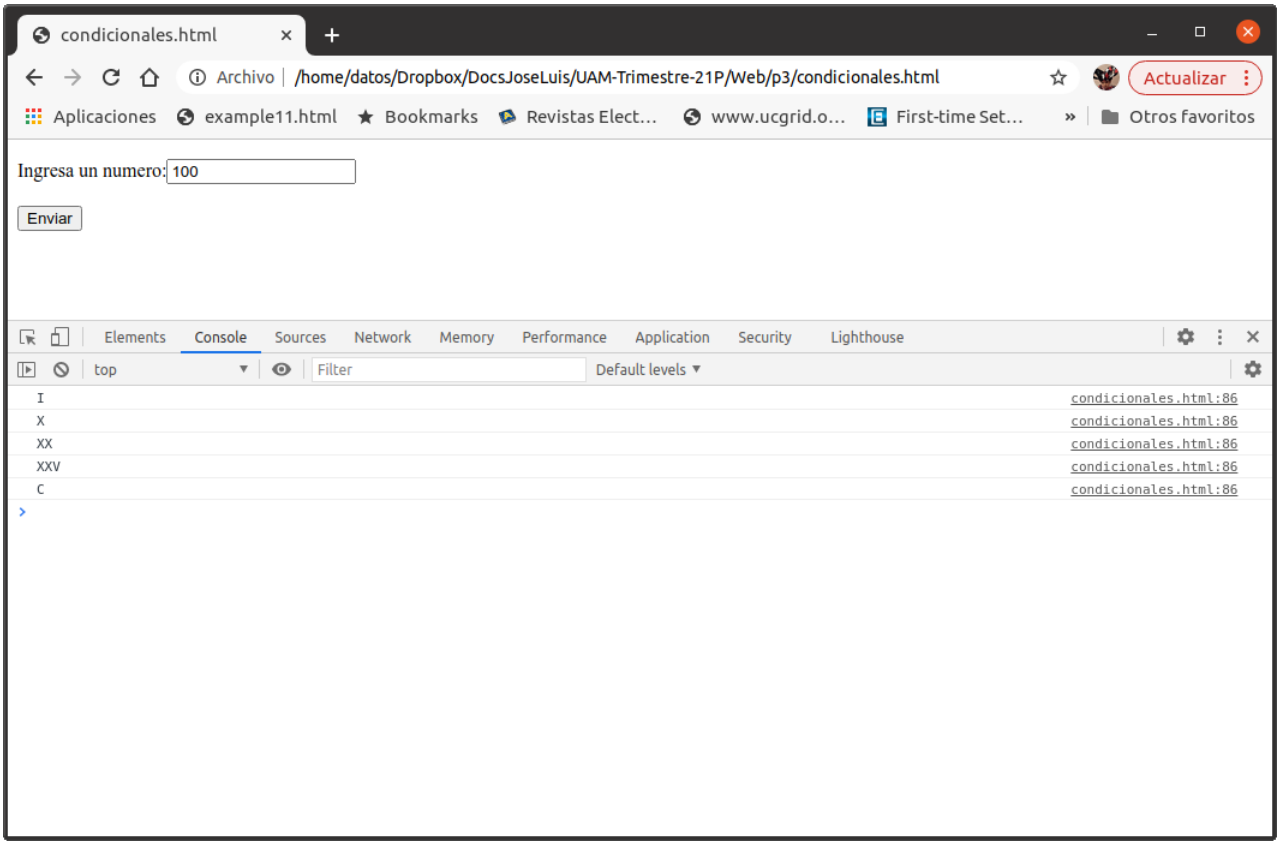
```
1   <!DOCTYPE html>
2   <html>
3      <body>
4         <p></p>
5         <script>
6
7         function getRomano(){
8            romano="";
9            str=document.getElementById("input").value;
10           valor=parseInt(str);
11
12           if(valor<0)
13               alert("El numero debe ser positivo");
14           else
15               if(valor>100)
16                  alert("El valor dene estar entre 1 y 100");
17               else{
18
19                    c=Math.floor(valor/10);
20                    r=valor%10;
21
22                    switch(c){
23                       case 0:
24                                 break;
25                       case 1:
26                                 romano="X";
27                                 break;
28                       case 2:
29                                 romano="XX";
30                                 break;
31                       case 3:
32                                 romano="XXX";
```

```
33                              break;
34                      case 4:
35                              romano="XL";
36                              break;
37                      case 5:
38                              romano="L";
39                              break;
40                      case 6:
41                              romano="LX";
42                              break;
43                      case 7:
44                              romano="LXX";
45                              break;
46                      case 8:
47                              romano="LXXX";
48                              break;
49                      case 9:
50                              romano="XC"
51                      default:
52                              romano="C"
53
54                  }
55                  switch(r){
56                      case 0:
57                              break;
58                      case 1:
59                              romano=romano +"I";
60                              break;
61                      case 2:
62                              romano=romano +"II";
63                              break;
64                      case 3:
65                              romano=romano +"III";
66                              break;
67                      case 4:
68                              romano=romano +"IV";
69                              break;
70                      case 5:
71                              romano=romano +"V";
72                              break;
73                      case 6:
74                              romano=romano +"VI";;
75                              break;
76                      case 7:
77                              romano=romano +"VII";
78                              break;
79                      case 8:
```

```
80                                romano=romano +"VIII";;
81                                break;
82                        default:
83                                romano=romano +"IX";
84
85                    }
86                    console.log(romano);
87
88                }
89            }
90
91            </script>
92
93        <form action="javascript:getRomano();">
94            Ingresa un numero:<input type="number" name="cajas" placeholder="p.e. 45"
                id="input" required/><br><br>
95            <input type="submit" value="Enviar">
96        </form>
97        </body>
98    </html>
```

Código 6: Ejemplo de condicionales

Salida:



1.5.2 Estructuras de repetición

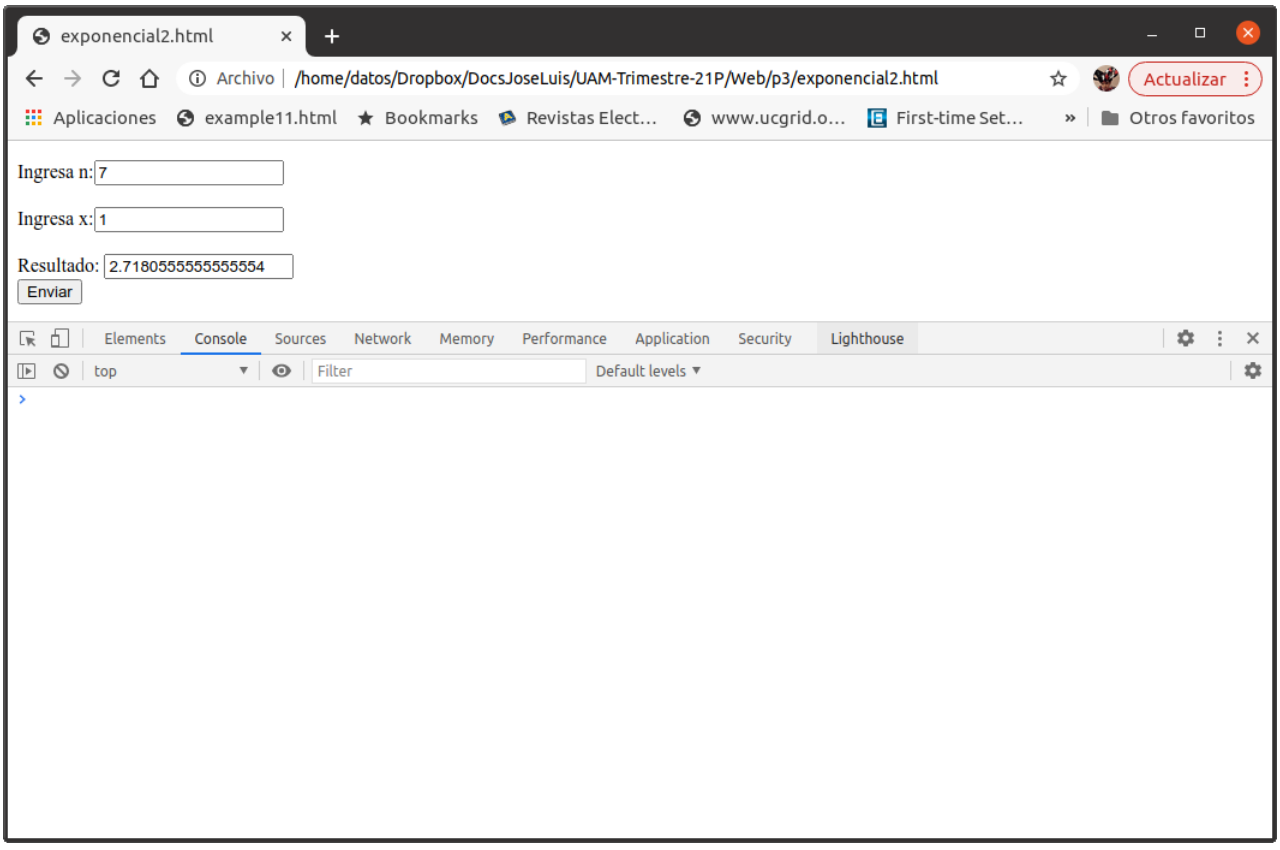
JavaScript soporta las estructuras **for**, **while** y **do-while** con la sintaxis similar al del lenguaje C y Java.

```
<!DOCTYPE html>
<html>
   <body>
      <p></p>
      <script>



         function getAproximacion(){

            str1=document.getElementById("input1").value;
            valor1=parseInt(str1);
            str2=document.getElementById("input2").value;
            valor2=parseInt(str2);

            r=aproximacion(valor1,valor2);

            document.getElementById("output").value=r;

         }

         function factorial(n){
            fact=1;
            for(i=1;i<=n;i++)
               fact=fact*i;
            return fact;

         }

         function potencia(base,exp){
            pot=1;
            while(exp>0){
               pot=pot*base;
               exp--;
            }
            return pot;

         }

          function aproximacion(n,x){
             suma=0;
             do{
```

```
43                 suma=suma+potencia(n,x)/factorial(n);
44                 n--;
45
46             }while(n>=0);
47             return suma;
48
49         }
50
51
52       </script>
53      <form action="javascript:getAproximacion();">
54       Ingresa n:<input type="number" name="cajas" placeholder="p.e. 45" id="
            input1" required/><br><br>
55       Ingresa x:<input type="number" name="cajas" placeholder="p.e. 45" id="
            input2" required/><br><br>
56       Resultado: <input type="text" name="cajas" value="" id="output" readonly><
            br>
57       <input type="submit" value="Enviar">
58    </form>
59    </body>
60 </html>
```

Código 7: Ejemplo de estructuras de repetición

Salida:



1.6 Ejercicio

Implementar una función que regrese **true** si todos los símbolos de una cadena se encuentran en otra cadena, **false** en otro caso. Los datos son leídos de cajas de texto y el resultado se despliega en una caja de texto.

1.7 Clases

1.7.1 Definición de una clase con class

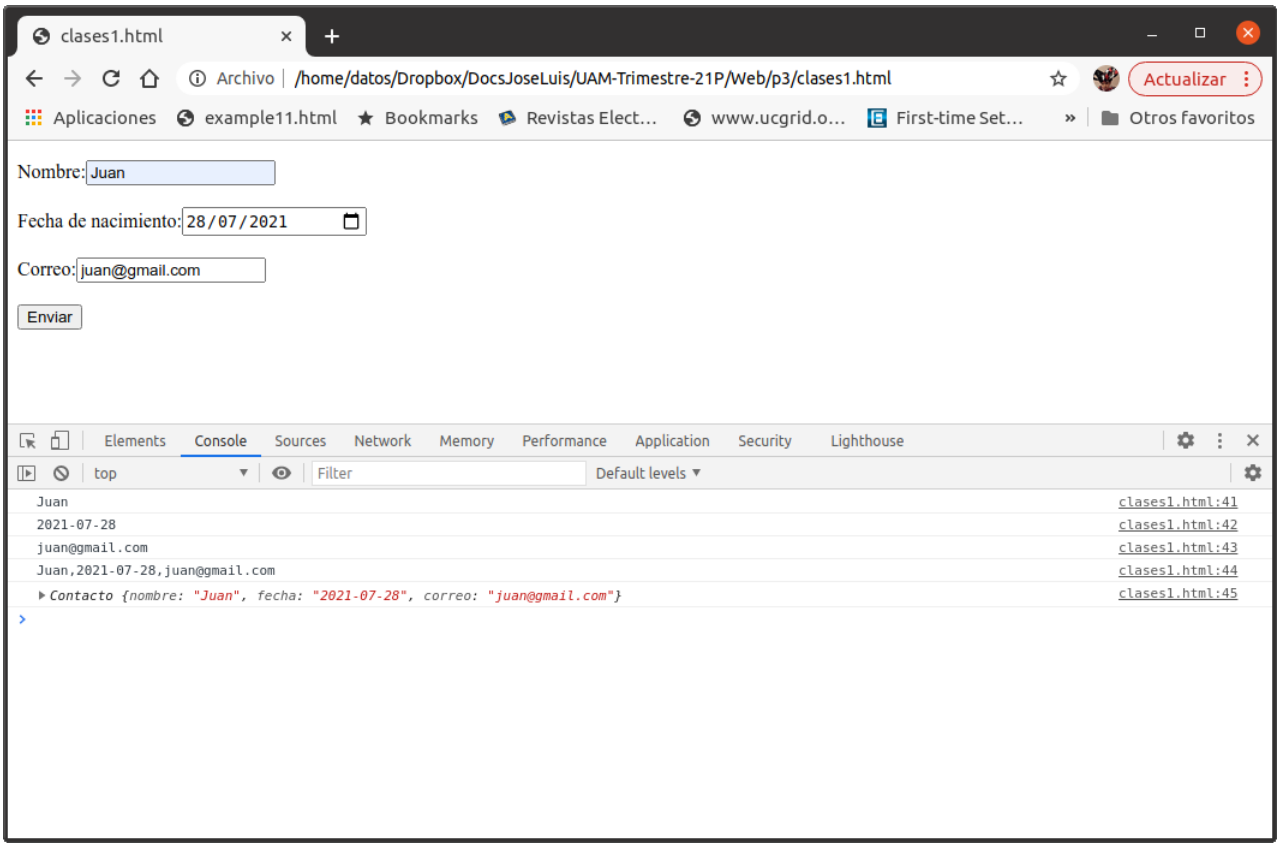
Las clases se pueden definir mediante la palabra **class**. La palabra **constructor** define el constructor de la clase, la palabra **this** permite establecer los atributos de la clase. Las palabras **get** y **set** definen atributos *setters* y *getters* para cambiar valores o bien obtener valores de los objetos creados. Al final de la clase de define un método llamado **toString**. Los objetos son creados mediante la palabra **new**.

```
1  <!DOCTYPE html>
2  <html>
3     <body>
4        <p></p>
5        <script>
6
7           var contacto;
8           class Contacto{
9               constructor(nombre, fecha,correo){
10                 this.nombre=nombre;
11                 this.fecha=fecha;
12                 this.correo=correo;
13              }
14              get getNombre(){
15                 return this.nombre;
16              }
17              get getFecha(){
18                 return this.fecha;
19              }
20              get getCorreo(){
21                 return this.correo;
22              }
23              set setNombre(nombre){
24                 this.nombre=nombre;
25              }
26              set setFecha(fecha){
27                 this.fecha=fecha;
28              }
29              set getCorreo(correo){
30                 this.correo=correo;
```

```
31              }
32              toString(){
33                return this.nombre+","+this.fecha+","+this.correo;
34              }
35          }
36          function getDatos(){
37             let form = document.getElementById('info');
38             let i;
39             contacto = new Contacto(form[0].value,form[1].value,form[2].value);
40
41             console.log(contacto.getNombre);
42             console.log(contacto.getFecha);
43             console.log(contacto.getCorreo);
44             console.log(contacto.toString());
45             console.log(contacto);
46
47          }
48
49       </script>
50    </body>
51    <form action="javascript:getDatos();" id="info">
52       Nombre:<input type="text" name="cajas" placeholder="p.e. Juan" id="
             input-nombre" required/><br><br>
53       Fecha de nacimiento:<input type="date" name="cajas" placeholder="p.e. 45"
             id="input-fecha" required/><br><br>
54       Correo:<input type="email" name="cajas" placeholder="juan@gmail.com" id="
             input-correo" required/><br><br>
55       <input type="submit" value="Enviar">
56    </form>
57 </html>
```

Código 8: Ejemplo de una clase

Salida:



1.7.2 Definición de una clase con function

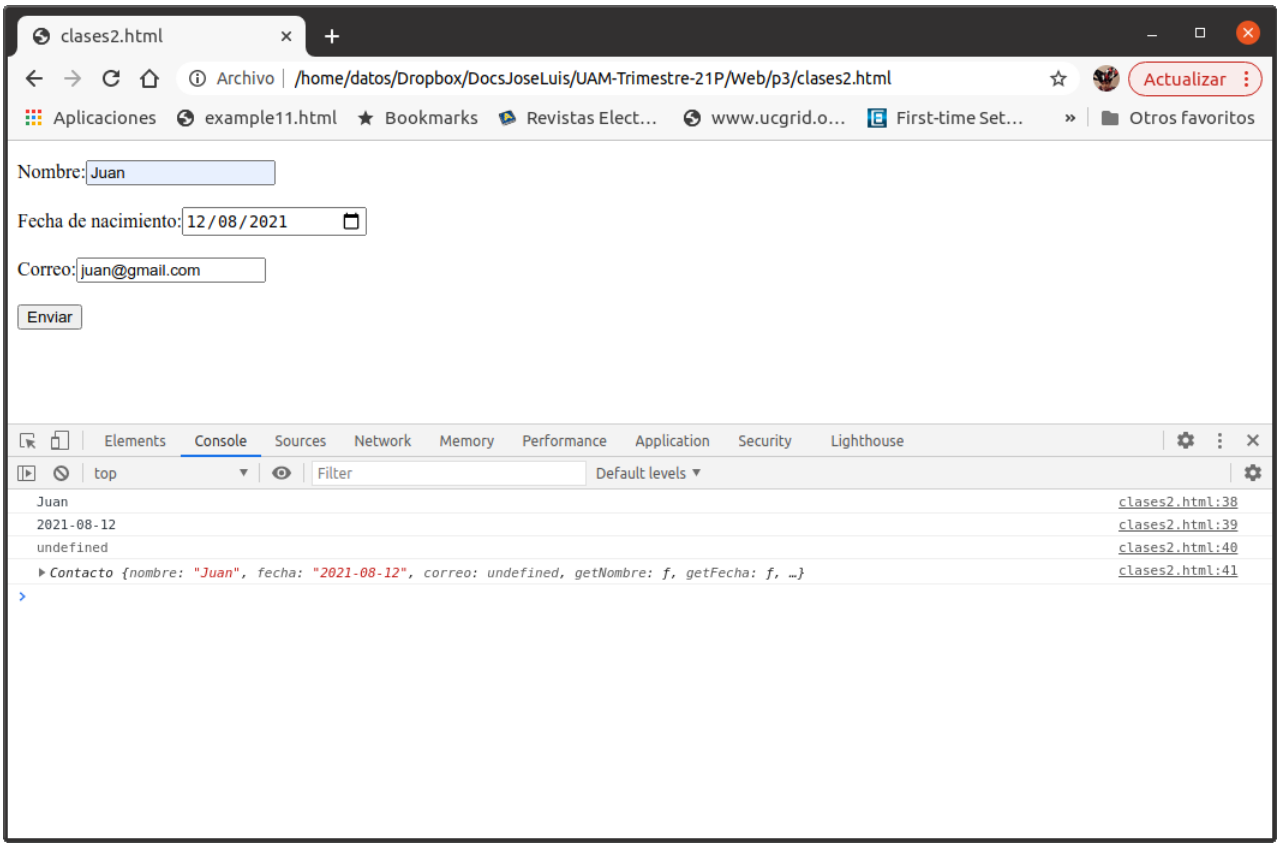
Es posible utilizar **function** para la declaración de una clase. Lo anterior se muestra en el siguiente ejemplo:

```
<!DOCTYPE html>
<html>
   <body>
      <p></p>
      <script>

         var contacto;
         function Contacto(nombre, fecha,correo){

             this.nombre=nombre;
             this.fecha=fecha;
             this.correo=correo;

             this.getNombre = function(){
                return this.nombre;
             }
             this.getFecha = function(){
                return this.fecha;
             }
             this.getCorreo = function(){
                return this.correo;
```

```
22              }
23              this.setNombre = function(nombre){
24                 this.nombre=nombre;
25              }
26              this.setFecha = function(fecha){
27                 this.fecha=fecha;
28              }
29              this.getCorreo = function(correo){
30                 this.correo=correo;
31              }
32          }
33          function getDatos(){
34             let form = document.getElementById('info');
35             let i;
36             contacto = new Contacto(form[0].value,form[1].value,form[2].value);
37
38             console.log(contacto.getNombre());
39             console.log(contacto.getFecha());
40             console.log(contacto.getCorreo());
41             console.log(contacto);
42
43          }
44
45       </script>
46    </body>
47    <form action="javascript:getDatos();" id="info">
48       Nombre:<input type="text" name="cajas" placeholder="p.e. Juan" id="
             input-nombre" required/><br><br>
49       Fecha de nacimiento:<input type="date" name="cajas" placeholder="p.e. 45"
             id="input-fecha" required/><br><br>
50       Correo:<input type="email" name="cajas" placeholder="juan@gmail.com" id="
             input-correo" required/><br><br>
51       <input type="submit" value="Enviar">
52    </form>
53 </html>
```

Código 9: Ejemplo de una clase con function

Salida:



1.7.3 Herencia

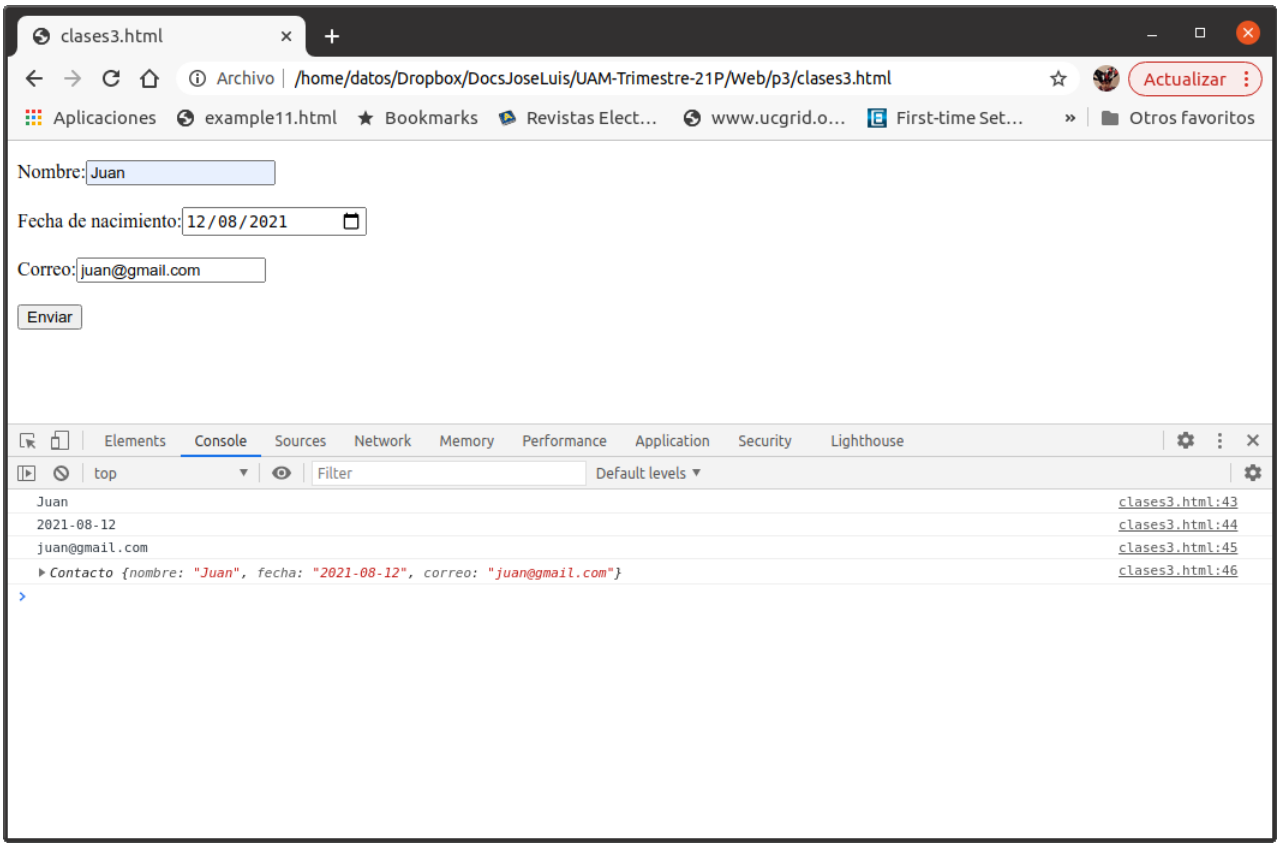
La herencia es posible mediante **extends**. La palabra **super** permite hacer referencia a la clase padre.

```
<!DOCTYPE html>
<html>
   <body>
      <p></p>
      <script>

         var contacto;
         class Persona{
            constructor(nombre, fecha){
                this.nombre=nombre;
                this.fecha=fecha;
             }
             get getNombre(){
                return this.nombre;
             }
             get getFecha(){
                return this.fecha;
             }
             set setNombre(nombre){
                this.nombre=nombre;
             }
```

```
22              set setFecha(fecha){
23                 this.fecha=fecha;
24              }
25          }
26          class Contacto extends Persona{
27              constructor(nombre, fecha,correo){
28                 super(nombre,fecha);
29                 this.correo=correo;
30              }
31              get getCorreo(){
32                 return this.correo;
33              }
34              set setCorreo(correo){
35                 this.correo=correo;
36              }
37          }
38          function getDatos(){
39             let form = document.getElementById('info');
40             let i;
41             contacto = new Contacto(form[0].value,form[1].value,form[2].value);
42
43             console.log(contacto.getNombre);
44             console.log(contacto.getFecha);
45             console.log(contacto.getCorreo);
46             console.log(contacto);
47
48          }
49
50       </script>
51    </body>
52    <form action="javascript:getDatos();" id="info">
53       Nombre:<input type="text" name="cajas" placeholder="p.e. Juan" id="
             input-nombre" required/><br><br>
54       Fecha de nacimiento:<input type="date" name="cajas" placeholder="p.e. 45"
             id="input-fecha" required/><br><br>
55       Correo:<input type="email" name="cajas" placeholder="juan@gmail.com" id="
             input-correo" required/><br><br>
56       <input type="submit" value="Enviar">
57    </form>
58 </html>
```

Código 10: Herencia

Salida:



1.7.4 Sobrecarga

La sobrecarga es posible aprovechando la palabra **undefined**. Se tiene que verificar que parámetros tienen valor **undefined** para la toma de decisiones del valor de los atributos.

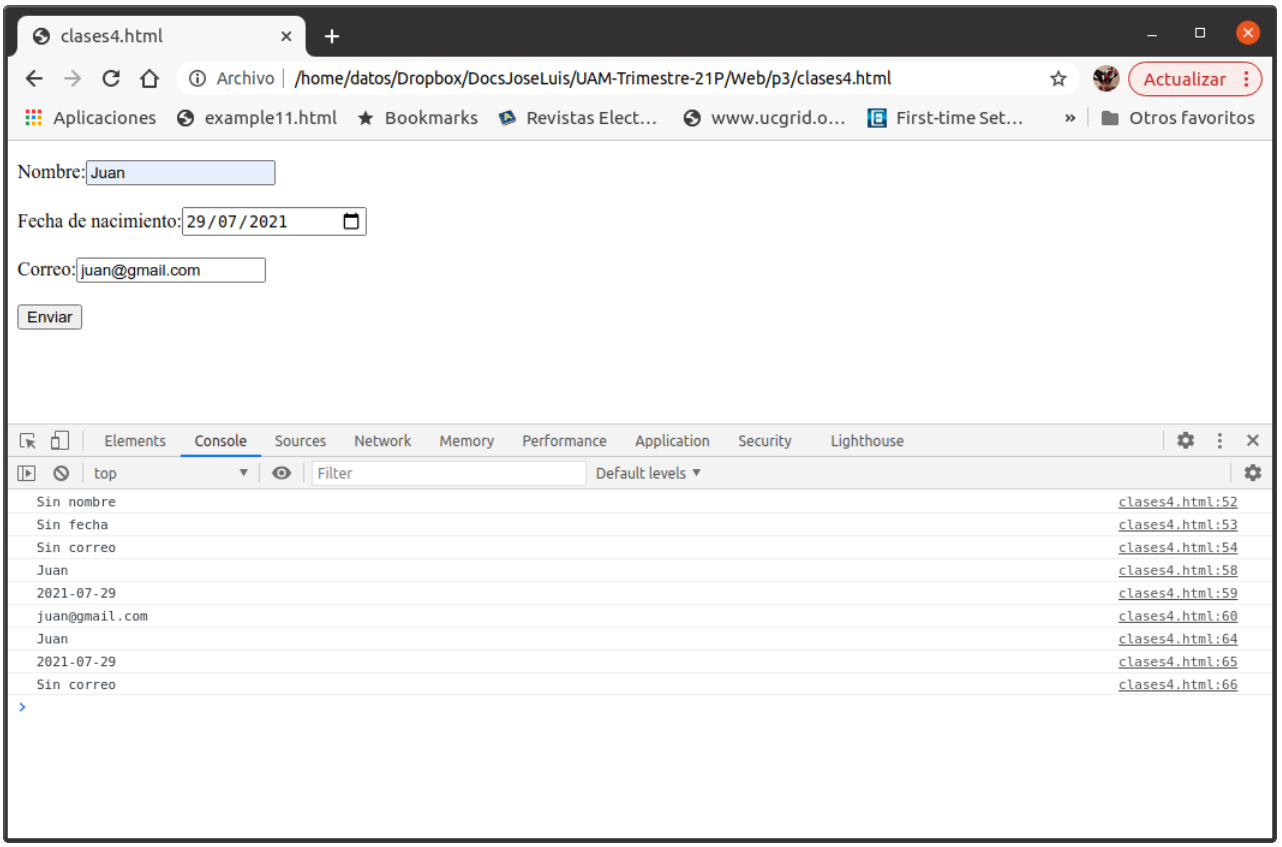
```
1  <!DOCTYPE html>
2  <html>
3     <body>
4        <p></p>
5        <script>
6
7           var contacto;
8           class Persona{
9              constructor(nombre, fecha){
10                this.nombre=nombre;
11                this.fecha=fecha;
12             }
13             get getNombre(){
14                return this.nombre;
15             }
16             get getFecha(){
17                return this.fecha;
18             }
19             set setNombre(nombre){
20                this.nombre=nombre;
21             }
```

```
22              set setFecha(fecha){
23                 this.fecha=fecha;
24              }
25          }
26          class Contacto extends Persona{
27              constructor(nombre, fecha,correo){
28                 if(correo==undefined){
29                    correo="Sin correo";
30                 }
31                 if(fecha==undefined){
32                    fecha="Sin fecha";
33                 }
34                 if(nombre==undefined){
35                    nombre="Sin nombre";
36                 }
37                 super(nombre,fecha);
38                 this.correo=correo;
39              }
40              get getCorreo(){
41                 return this.correo;
42              }
43              set getCorreo(correo){
44                 this.correo=correo;
45              }
46          }
47          function getDatos(){
48             let form = document.getElementById('info');
49             let i;
50             contacto = new Contacto();
51
52             console.log(contacto.getNombre);
53             console.log(contacto.getFecha);
54             console.log(contacto.getCorreo);
55
56             contacto = new Contacto(form[0].value,form[1].value,form[2].value);
57
58             console.log(contacto.getNombre);
59             console.log(contacto.getFecha);
60             console.log(contacto.getCorreo);
61
62             contacto = new Contacto(form[0].value,form[1].value);
63
64             console.log(contacto.getNombre);
65             console.log(contacto.getFecha);
66             console.log(contacto.getCorreo);
67          }
68
```

```
69        </script>
70     </body>
71     <form action="javascript:getDatos();" id="info">
72        Nombre:<input type="text" name="cajas" placeholder="p.e. Juan" id="
              input-nombre"/><br><br>
73        Fecha de nacimiento:<input type="date" name="cajas" placeholder="p.e. 45"
              id="input-fecha"/><br><br>
74        Correo:<input type="email" name="cajas" placeholder="juan@gmail.com" id="
              input-correo"/><br><br>
75        <input type="submit" value="Enviar">
76     </form>
77  </html>
```

Código 11: Sobrecarga

Salida:



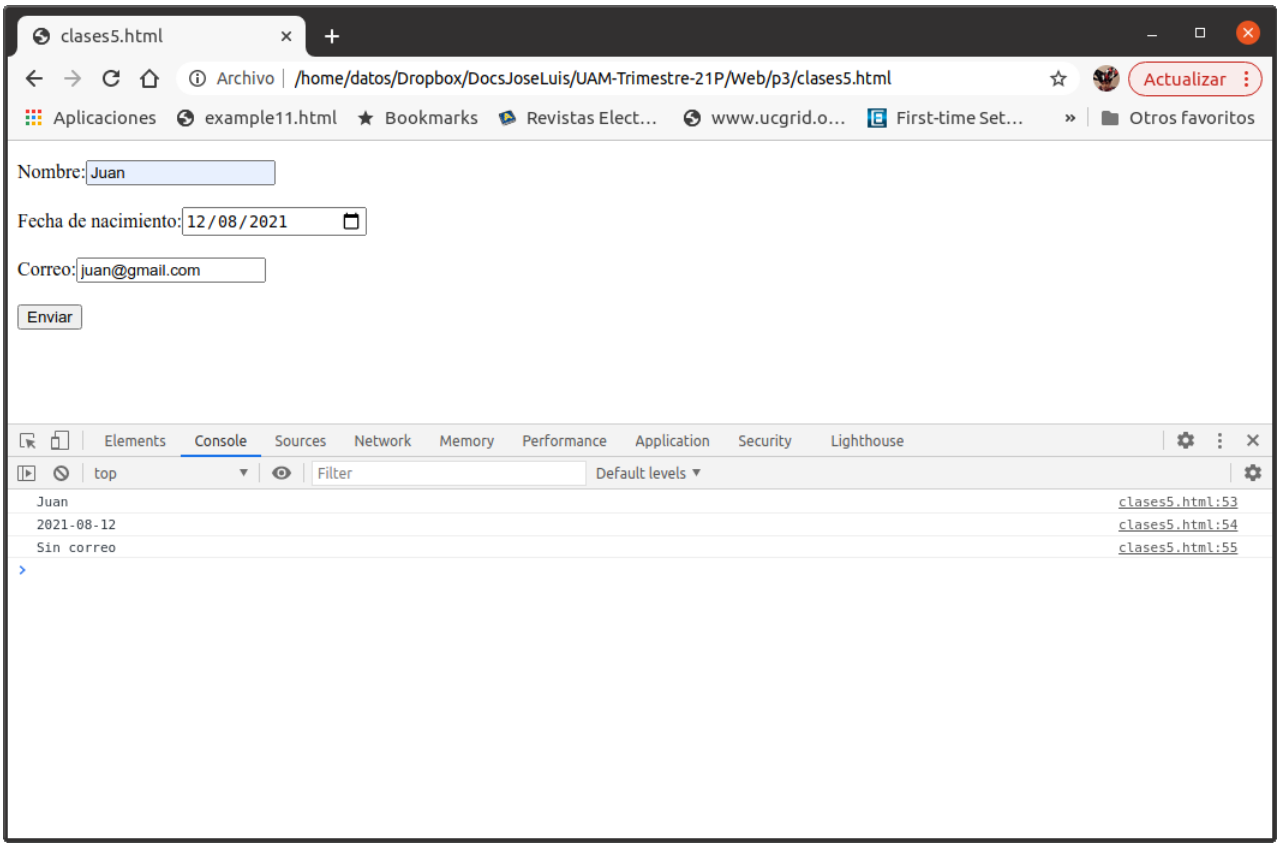

```
1  <!DOCTYPE html>
2  <html>
3     <body>
4        <p></p>
5        <script>
6
7           var contacto;
8           class Persona{
9              constructor(nombre, fecha){
```

```
10                this.nombre=nombre;
11                this.fecha=fecha;
12            }
13            get getNombre(){
14                return this.nombre;
15            }
16            get getFecha(){
17                return this.fecha;
18            }
19            set setNombre(nombre){
20                this.nombre=nombre;
21            }
22            set setFecha(fecha){
23                this.fecha=fecha;
24            }
25        }
26        class Contacto extends Persona{
27            constructor(nombre="Sin nombre", fecha="Sin fecha",correo="Sin
                  correo"){
28
29                super(nombre,fecha);
30                this.correo=correo;
31            }
32            get getCorreo(){
33                return this.correo;
34            }
35            set getCorreo(correo){
36                this.correo=correo;
37            }
38        }
39        function getDatos(){
40            let form = document.getElementById('info');
41            let i;
42
43            contacto = new Contacto(form[0].value,form[1].value);
44
45            console.log(contacto.getNombre);
46            console.log(contacto.getFecha);
47            console.log(contacto.getCorreo);
48        }
49
50    </script>
51  </body>
52  <form action="javascript:getDatos();" id="info">
53      Nombre:<input type="text" name="cajas" placeholder="p.e. Juan" id="
            input-nombre"/><br><br>
54      Fecha de nacimiento:<input type="date" name="cajas" placeholder="p.e. 45"
```

```
            id="input-fecha"/><br><br>
55        Correo:<input type="email" name="cajas" placeholder="juan@gmail.com" id="
              input-correo"/><br><br>
56        <input type="submit" value="Enviar">
57    </form>
58 </html>
```

Código 12: Sobrecarga inicializando los parámetros

Salida:



1.8 Document Object Model (DOM)

El DOM es un estándar W3C (World Wide Web Consortium) para acceder a documentos:

”The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.”

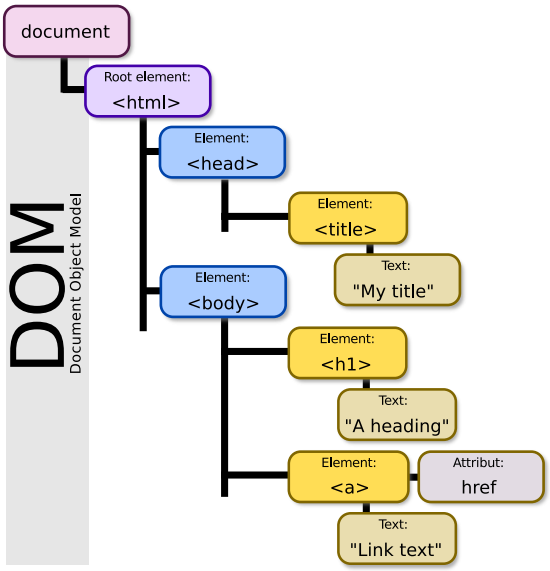
El estándar W3C DOM es separado en tres partes diferentes:

1. Core DOM - modelo estándar para todo tipo de documentos.
2. XML DOM - modelo estándar para documentos XML.
3. HTML DOM - modelo estándar para documentos HTML.

Cuando una página se carga, el navegador crea un documento de modelo de objetos. El modelo DOM se construye como un árbol de objetos. Con el modelo de objetos, JavaScript puede crear páginas HTML dinámicas.

¿Qué operaciones puede realizar JS con los elementos del DOM?

1. JavaScript puede cambiar todos los elementos HTML en la página
2. JavaScript puede cambiar todos los atributos HTML en la página
3. JavaScript puede cambiar todos los estilos CSS en la página
4. JavaScript puede eliminar elementos y atributos HTML
5. JavaScript puede agregar nuevos elementos HTML y atributos
6. JavaScript puede escuchar de todos los eventos en la página
7. JavaScript puede crear nuevos eventos HTMLen la página.



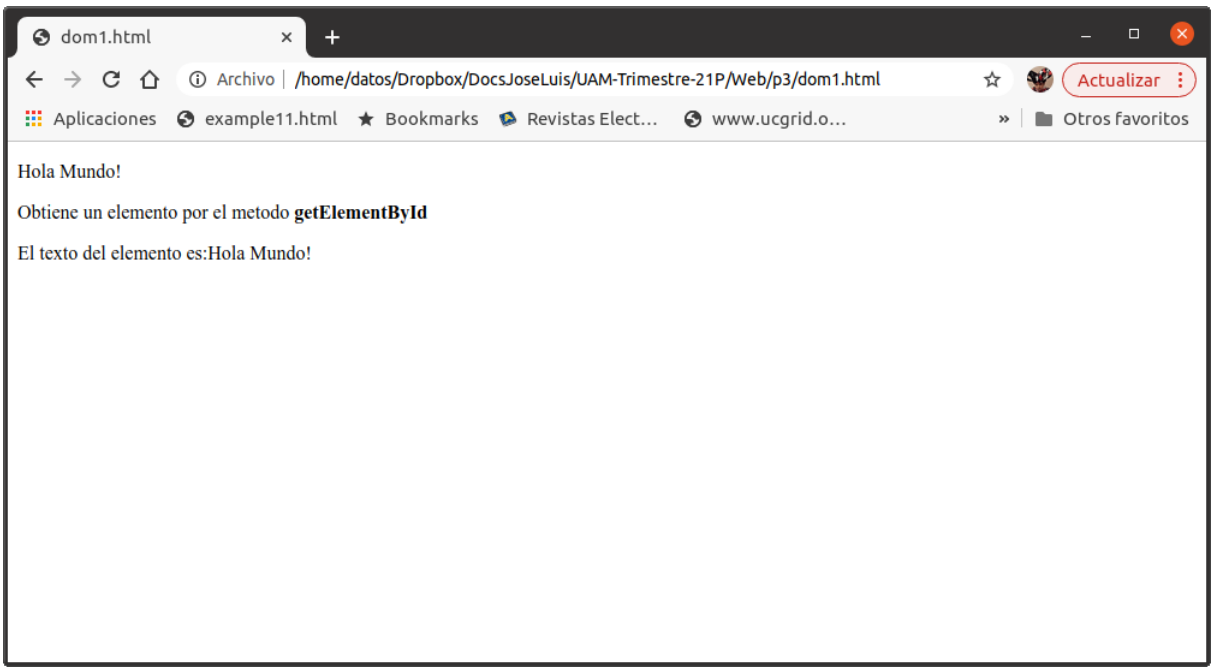
1.8.1 Encontrando elemento por id

```
<!DOCTYPE html>
<html>
   <body>

      <p id="intro">Hola Mundo!</p>
      <p>Obtiene un elemento por el metodo <b>getElementById</b></p>
      <p id="demo"></p>
      <script>
         var elemento = document.getElementById("intro");
```

```
10          document.getElementById("demo").innerHTML = "El texto del elemento es:"
                + elemento.innerHTML;
11      </script>
12
13   </body>
14 </html>
```

Código 13: Encontrar elemento por id

Salida:

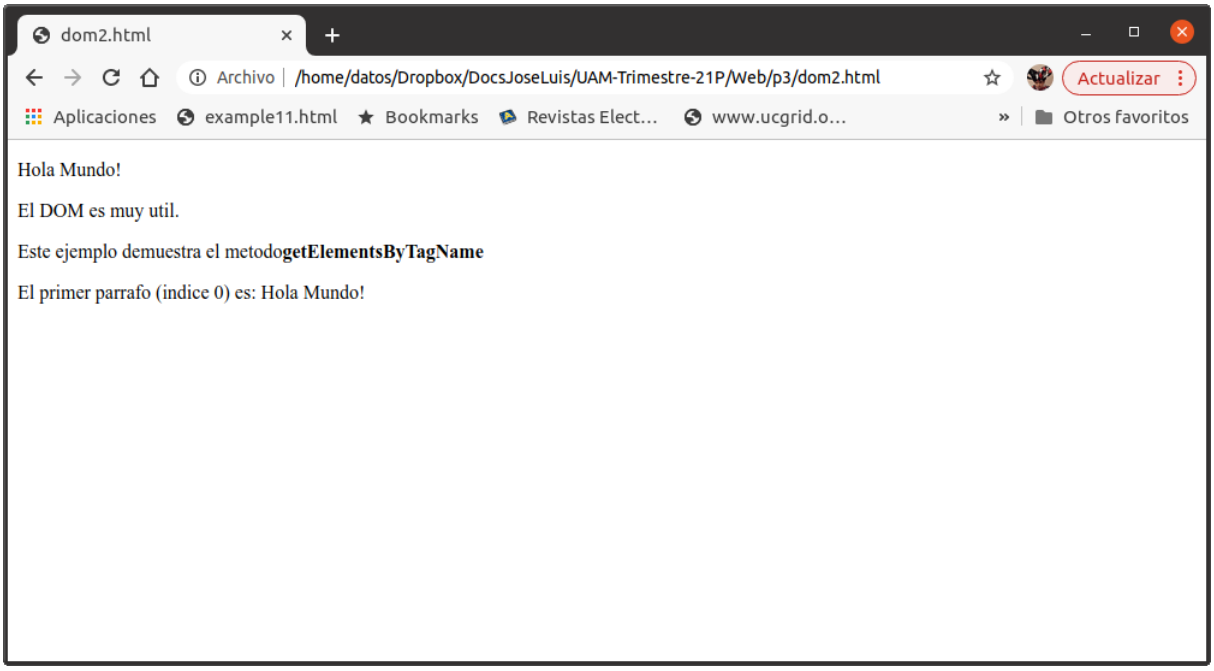


1.8.2 Encontrando elementos por nombre de etiqueta

```
1  <!DOCTYPE html>
2  <html>
3     <body>
4        <p>Hola Mundo!</p>
5        <p>El DOM es muy util.</p>
6        <p>Este ejemplo demuestra el metodo<b>getElementsByTagName</b> </p>
7
8        <p id="demo"></p>
9
10       <script>
11          var x = document.getElementsByTagName("p");
12          document.getElementById("demo").innerHTML = 'El primer parrafo (indice
                0) es: ' + x[0].innerHTML;
13       </script>
14    </body>
15 </html>
```

Código 14: Encontrar elemento por etiqueta

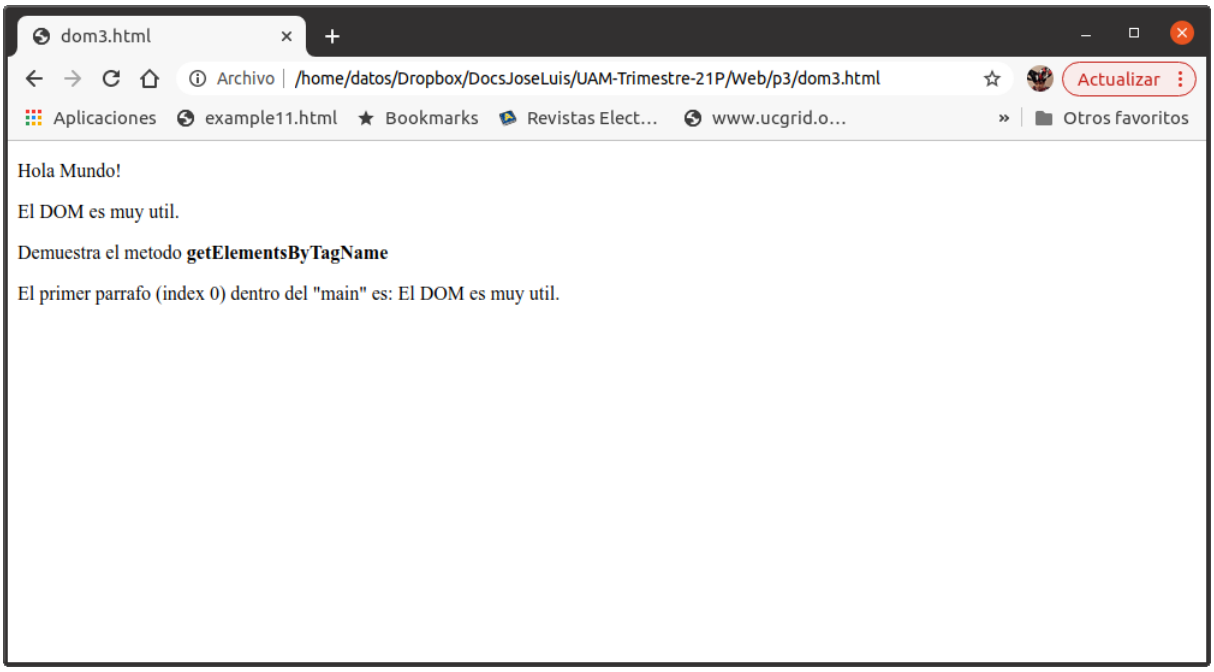
Salida:




```
<!DOCTYPE html>
<html>
   <body>
      <p>Hola Mundo!</p>
      <div id="main">
         <p>El DOM es muy util.</p>
         <p>Demuestra el metodo <b>getElementsByTagName</b></p>
      </div>
      <p id="demo"></p>
      <script>
         var x = document.getElementById("main");
         var y = x.getElementsByTagName("p");
         document.getElementById("demo").innerHTML = 'El primer parrafo (index
            0) dentro del "main" es: ' + y[0].innerHTML;
      </script>
   </body>
</html>
```

Código 15: Encontrar elemento por etiqueta

Salida:

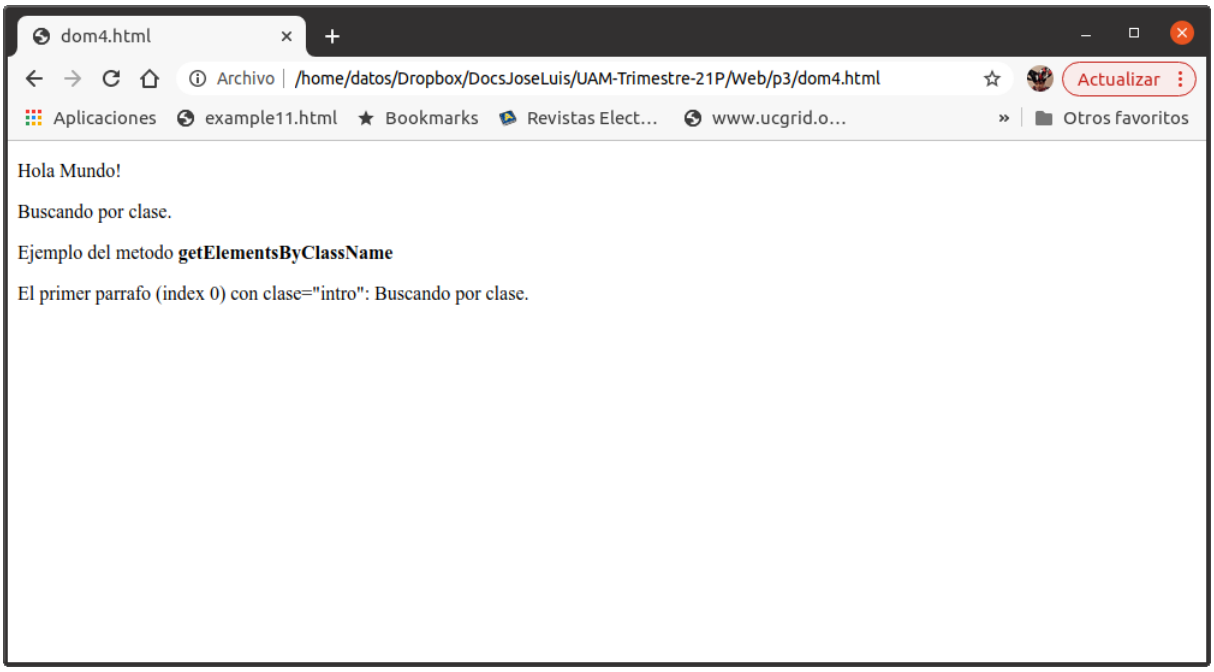


1.8.3 Encontrando elementos por nombre de clase

```
<!DOCTYPE html>
<html>
   <body>

      <p>Hola Mundo!</p>
      <p class="intro">Buscando por clase.</p>
      <p class="intro">Ejemplo del metodo <b>getElementsByClassName</b> </p>
      <p id="demo"></p>
      <script>
         var x = document.getElementsByClassName("intro");
         document.getElementById("demo").innerHTML = 'El primer parrafo (index
            0) con clase="intro": ' + x[0].innerHTML;
      </script>
   </body>
</html>
```

Código 16: Encontrar elemento por clase

Salida:

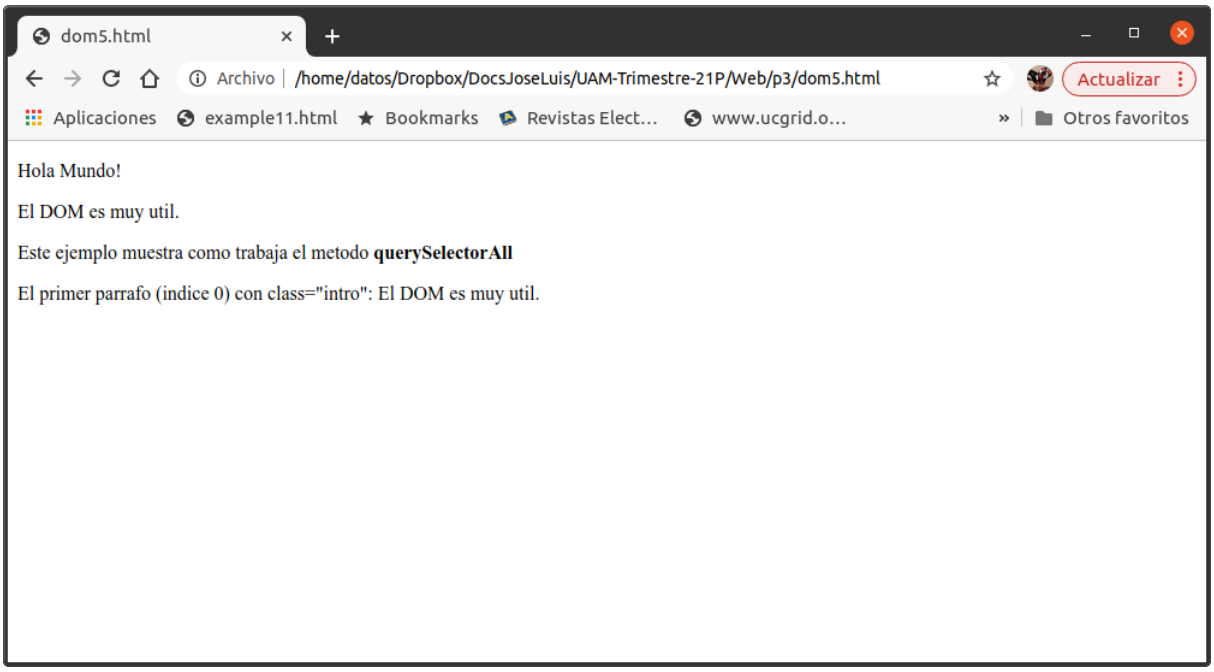


1.8.4 Encontrando elementos selector CSS

```
<!DOCTYPE html>
<html>
   <body>
      <p>Hola Mundo!</p>
      <p class="intro">El DOM es muy util.</p>
      <p class="intro">Este ejemplo muestra como trabaja el metodo <b>
         querySelectorAll</b> </p>
      <p id="demo"></p>
      <script>
         var x = document.querySelectorAll("p.intro");
         document.getElementById("demo").innerHTML = 'El primer parrafo (indice
            0) con class="intro": ' + x[0].innerHTML;
     </script>

   </body>
</html>
```

Código 17: Encontrar elemento por selector

Salida:



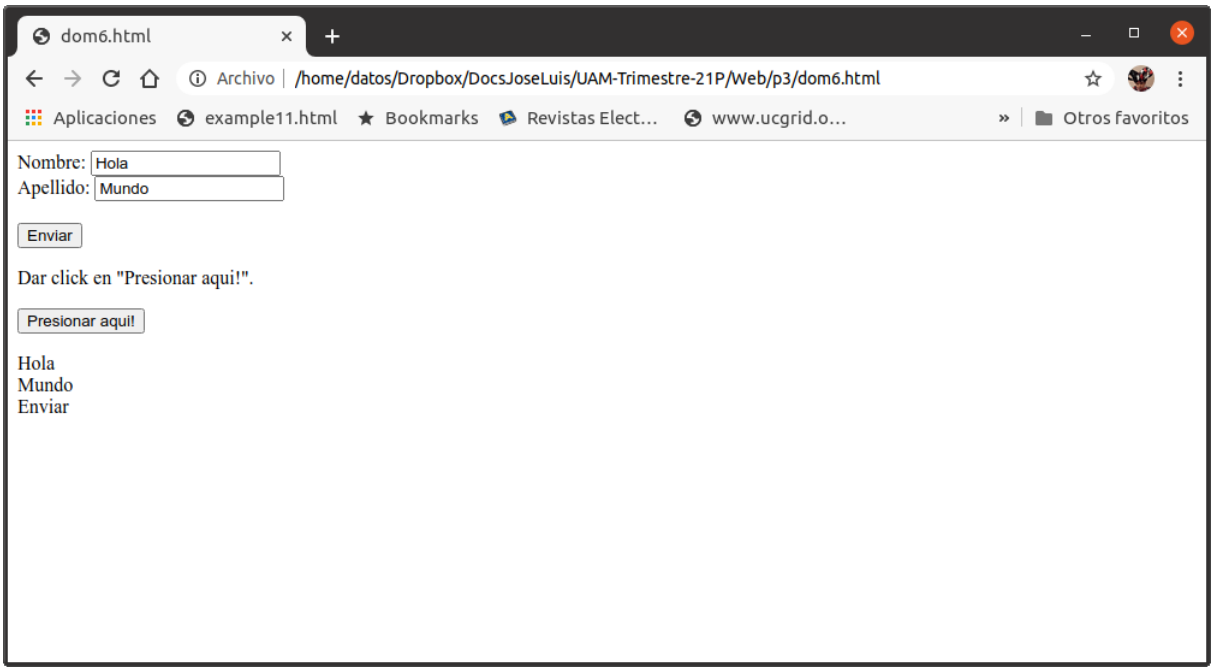
1.8.5 Encontrando elementos de una colección form

```
<!DOCTYPE html>
<html>
<body>

<form id="frm1">
  Nombre: <input type="text" name="fnombre" value="Hola"><br>
  Apellido: <input type="text" name="lnombre" value="Mundo"><br><br>
  <input type="submit" value="Enviar">
</form>

<p>Dar click en "Presionar aqui!".</p>

<button onclick="getDatos()">Presionar aqui!</button>

<p id="demo"></p>

<script>
function getDatos() {
    var x = document.forms["frm1"];
    var text = "";
    var i;
    for (i = 0; i < x.length ;i++) {
        text += x.elements[i].value + "<br>";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

```

```
</body>
</html>
```

Código 18: Encontrar elementos de un formulario

Salida:



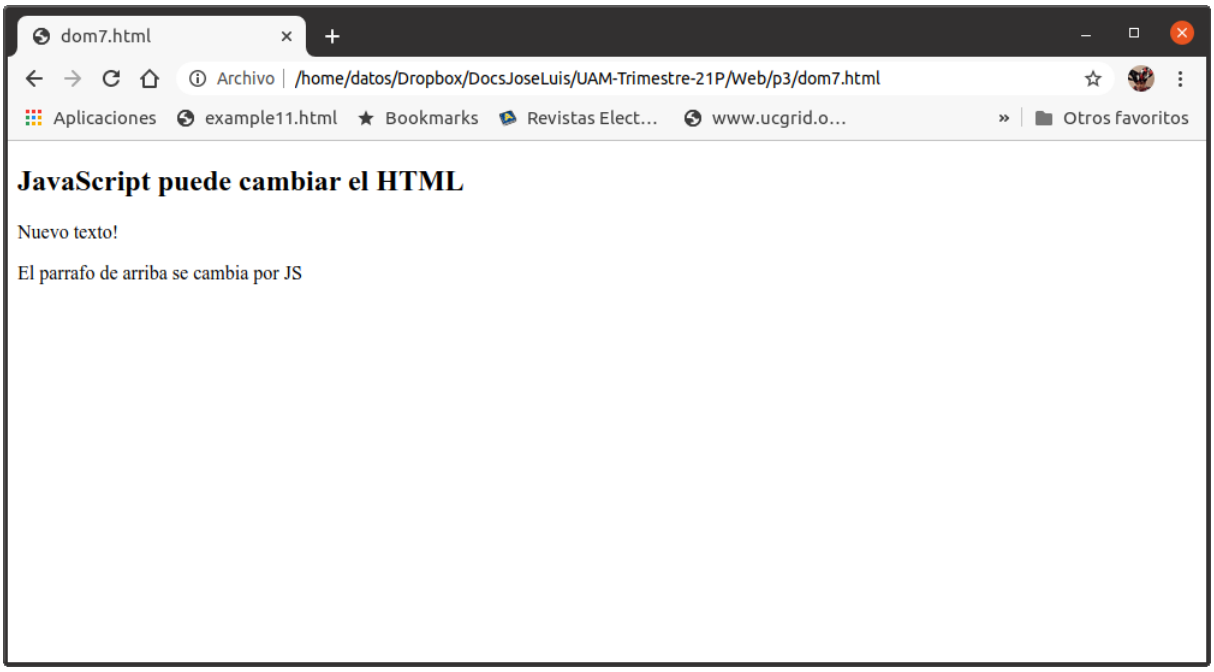
1.8.6 Cambiando el contenido de un elemento HTML

La manera más fácil para modificar el contenido de un elemento HTML es usando la propiedad **innerHTML**.

```
<!DOCTYPE html>
<html>
<body>

   <h2>JavaScript puede cambiar el HTML</h2>

   <p id="p1">Hola Mundo!</p>

   <p>El parrafo de arriba se cambia por JS</p>

   <script>
      document.getElementById("p1").innerHTML = "Nuevo texto!";
   </script>
</body>
</html>
```

Código 19: Modificando contenido de un elemento HTML

Salida:



1.8.7 Cambiando el valor de un atributo

Para cambiar el valor de un atributo HTML se usa la sintaxis:

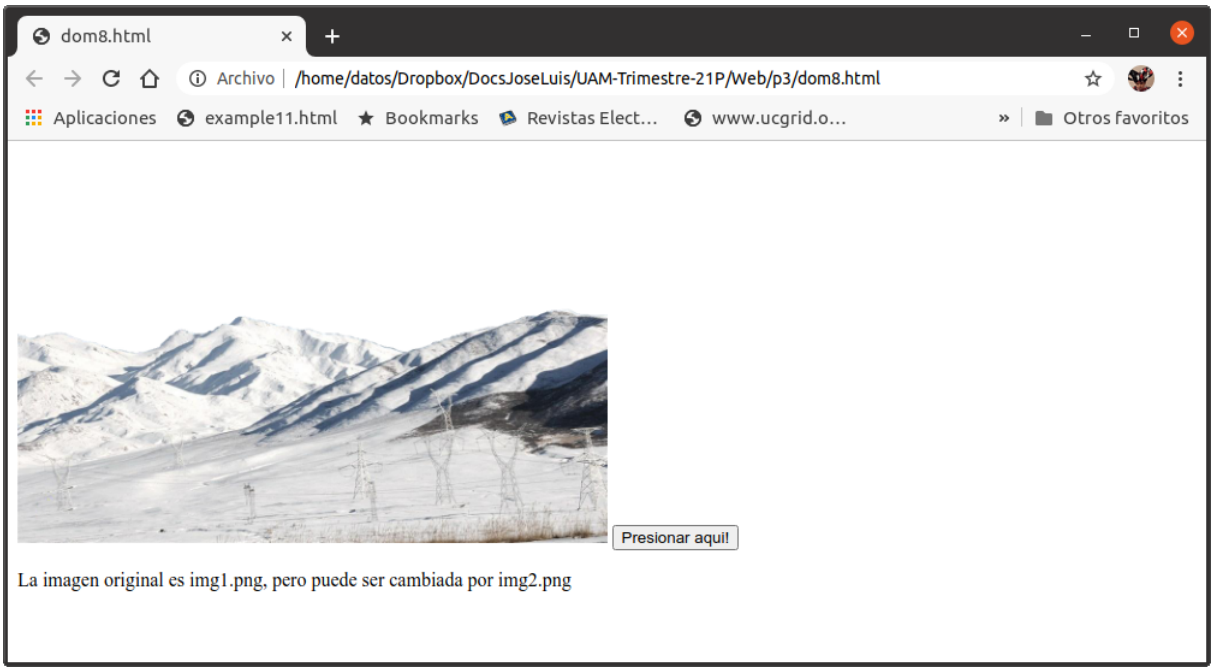
document.getElementById(id).attribute = nuevo valor

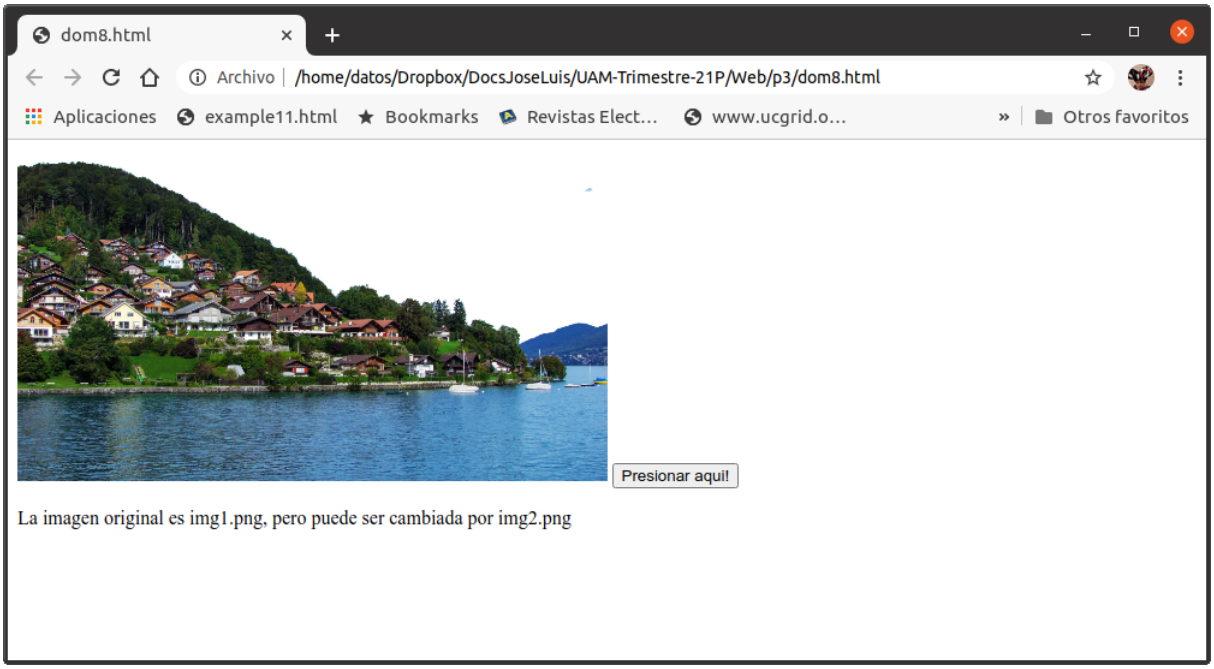
En el siguiente ejemplo se cambia el valor del atributo src de un elemento <**img**>.

```
<!DOCTYPE html>
<html>
<body>

   <img id="image" src="img1.png" width="50%">
   <button onclick="setCambiar()">Presionar aqui!</button>

   <script>

   function setCambiar() {
      document.getElementById("image").src = "img2.png";
   }
   </script>

   <p>La imagen original es img1.png, pero puede ser cambiada por img2.png</p>

</body>
</html>
```

Código 20: Modificando el atributo de un elemento HTML

Salida:





Nota: para este ejemplo buscar dos imágenes y renombrarlas img1.png y img2.png respectivamente.

1.8.8 Cambiando un estilo

Para cambiar el estilo de un elemento HTML se usa la sintaxis:

document.getElementById(id).style.property = new style

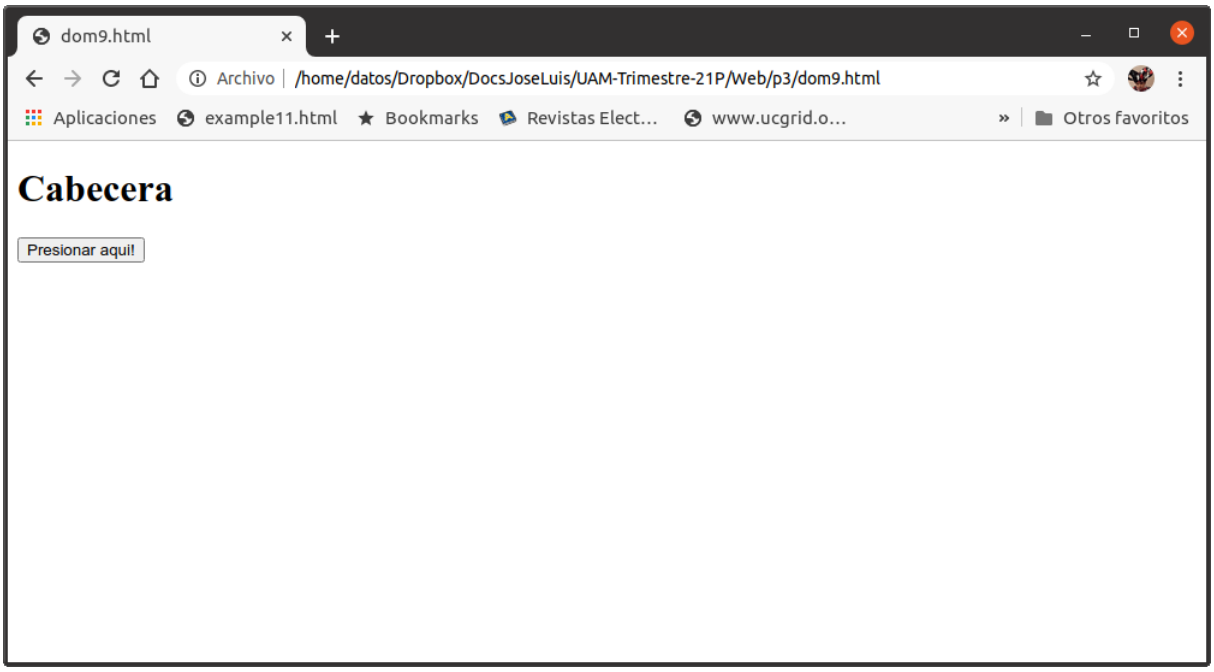
En el siguiente ejemplo se cambia el estilo color del elemento identificado por **id1**.

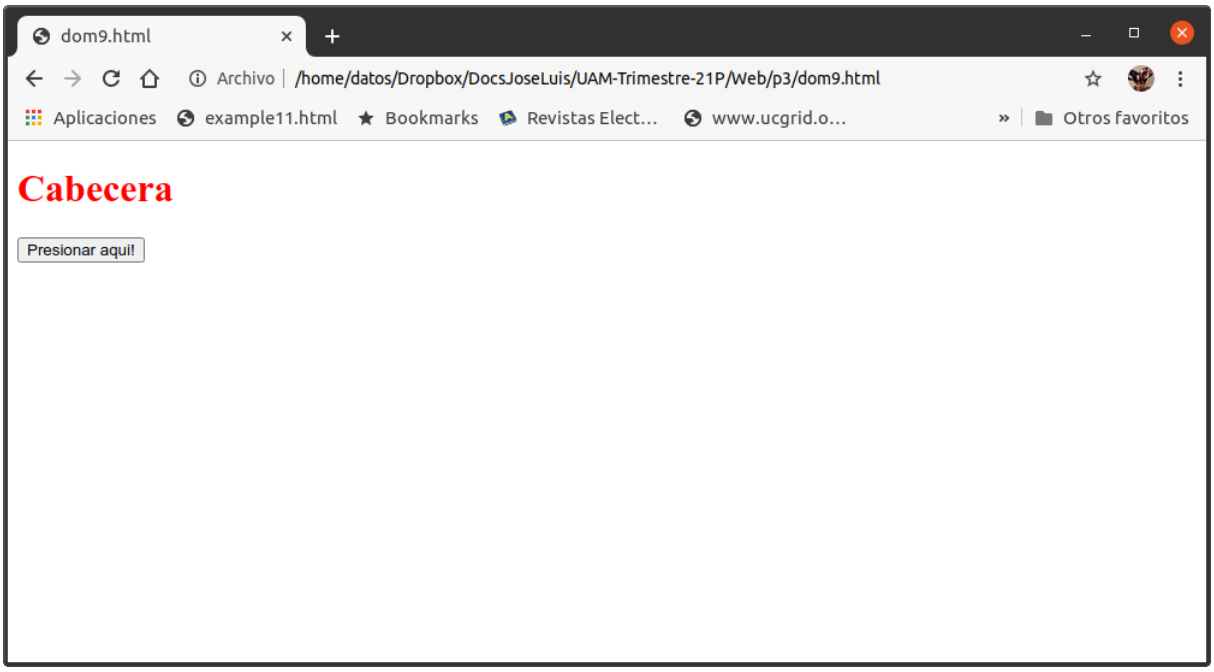
```
1 <!DOCTYPE html>
2 <html>
```

```
3  <body>
4
5     <h1 id="id1">Cabecera</h1>
6
7     <button type="button" onclick="setCambiar();">Presionar aqui!</button>
8     <script>
9
10    function setCambiar() {
11       document.getElementById('id1').style.color = 'red';
12    }
13    </script>
14 </body>
15 </html>
```

Código 21: Modificando el estilo de un elemento HTML

Salida:

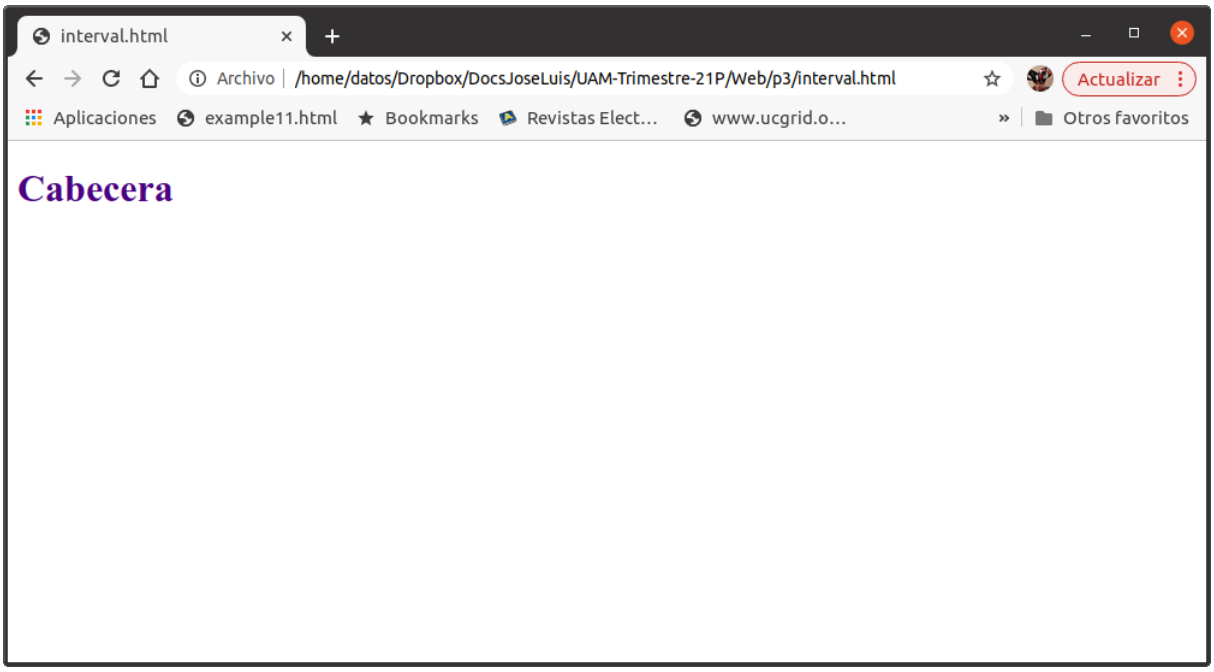





```
<!DOCTYPE html>
<html>
<body>

   <h1 id="id1">Cabecera</h1>


   <script>
   var color = ["Blue ", "Green", "Red", "Orange", "Violet", "Indigo", "Yellow "
       ];
   var i=0;
   function getRandomInt(max) {
     return Math.floor(Math.random() * max);
   }
   function setCambiar() {
      document.getElementById('id1').style.color = color[(i+1)%color.length];
      i++;
   }
   setInterval(setCambiar, 3000);
   </script>
</body>
</html>
```

Código 22: Modificando el estilo de un elemento HTML

Salida:



1.8.9 Ejercicio: Reloj

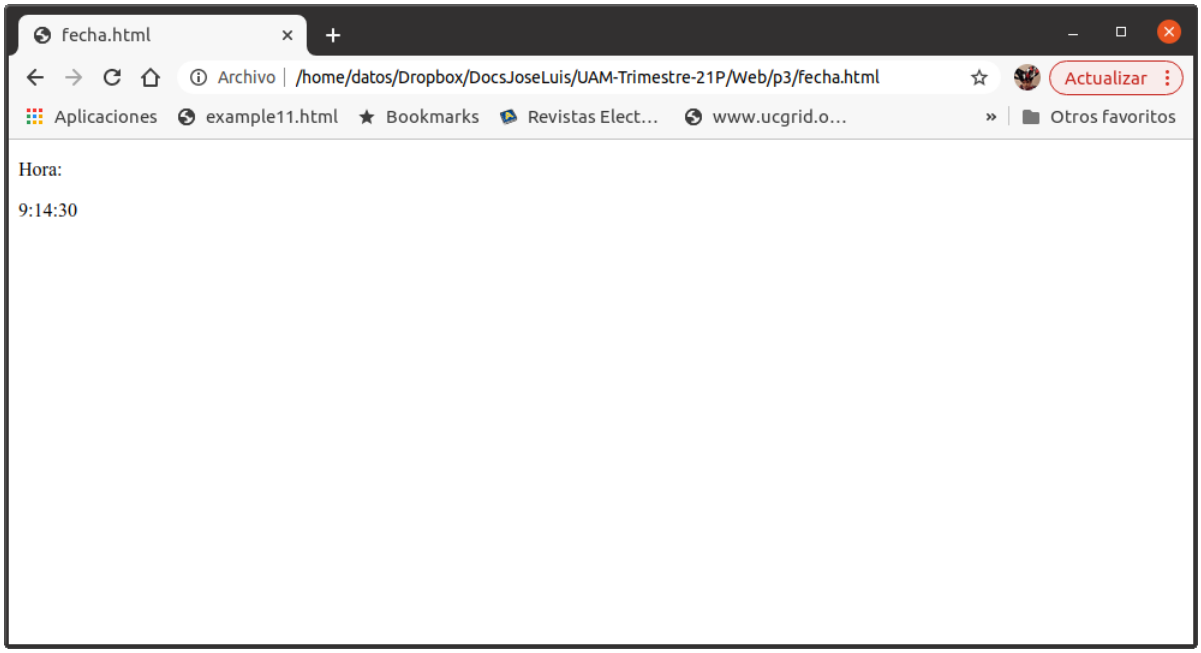
Implemente un reloj usando HTML, CSS y JavaScript. El segundero del reloj se debe mover de forma constante. Considere el siguiente código de ayuda.

```
<!DOCTYPE html>
<html>
<body>

<p>Hora:</p>


<p id="test"></p>

<script>

    var d = new Date();
    var dia = d.getDate();
    var hora = d.getHours();
    var minuto = d.getMinutes();
    var segundo = d.getSeconds();


    document.getElementById("test").innerHTML=hora+":"+minuto+":"+segundo;



</script>

</body>
```

```
26 </html>
```

Código 23: Objeto Date en JavaScript

Salida:



Reloj esperado:

1.9 Eventos del teclado y del ratón

Por medio de JavaScript es posible controlar varios eventos del ratón, del teclado y del touch.

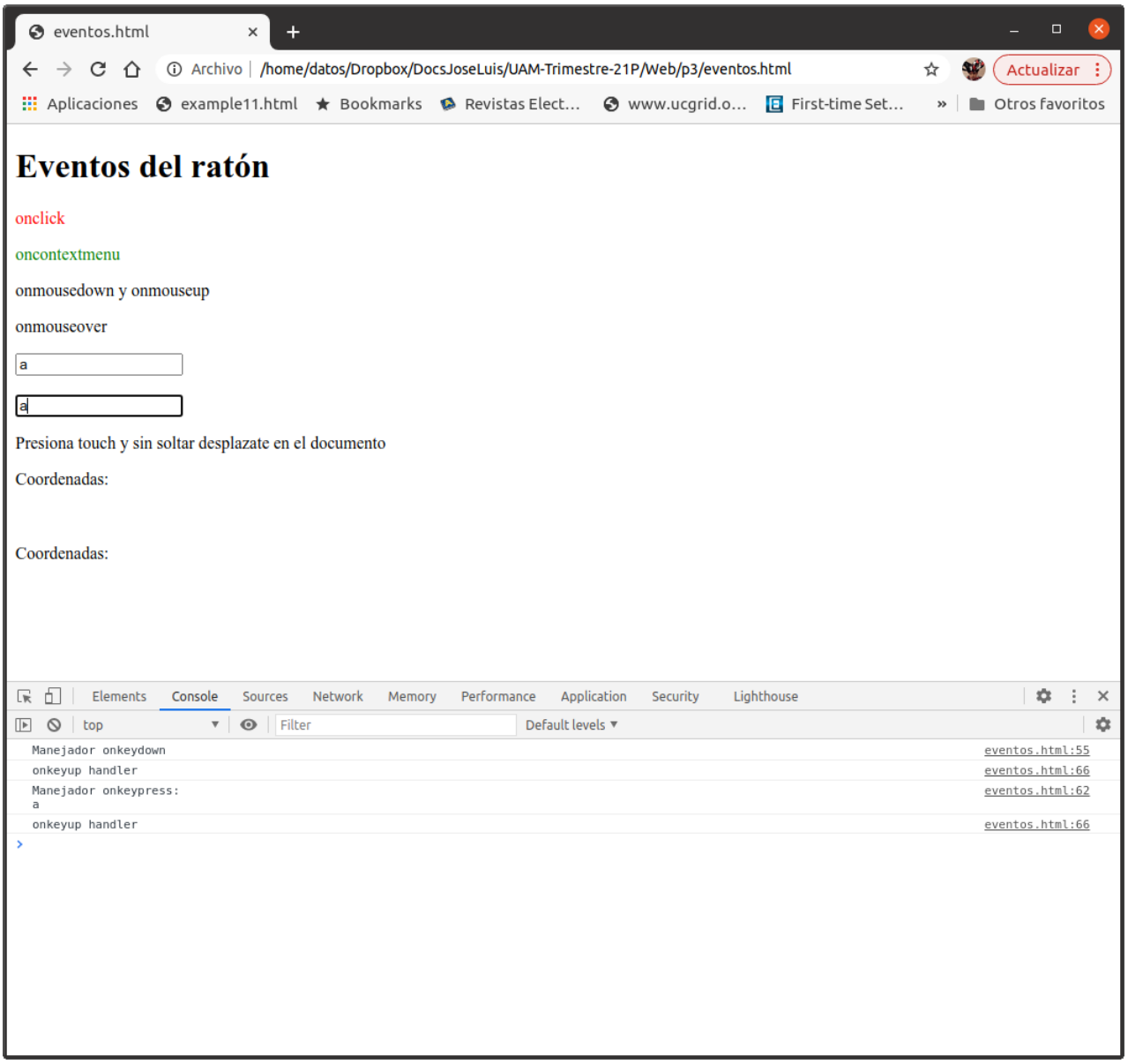
```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h1>Eventos del ratón</h1>
6
7 <p onclick="manejadorOnclik()" id="evento1">onclick</p>
8
9 <p oncontextmenu="manejadorOncontextMenu()" id="evento2">oncontextmenu</p>
```

```
10
11 <p onmousedown="manejadorMouseDown()" onmouseup="manejadorMouseUp()" id="evento3
       ">onmousedown y onmouseup</p>
12
13 <p onmouseover="manejadorOnmouseover()" onmouseout="manejadorOnmouseout()" id="
       evento4">onmouseover</p>
14
15 <input type="text" onkeydown="manejadorOnkeydown(event)" onkeyup="
       manejadorOnkeyup(event)" placeholder="onkeydown">
16
17 </br>
18 </br>
19
20 <input type="text" onkeypress="manejadorOnkeypress(event)" onkeyup="
       manejadorOnkeyup(event)" placeholder="onkeypress">
21
22 <p ontouchmove="manejadorOntouchmove(event)">Presiona touch y sin soltar
       desplazate en el documento</p>
23
24 Coordenadas:<p id="test"></p>
25 </br>
26 </br>
27 Coordenadas:<p id="test2"></p>
28
29 <script>
30 function manejadorOnclik() {
31    document.getElementById("evento1").style.color = "red";
32 }
33
34 function manejadorOncontextMenu() {
35    document.getElementById("evento2").style.color = "green";
36 }
37
38 function manejadorMouseDown() {
39   document.getElementById("evento3").style.color = "lime";
40 }
41
42 function manejadorMouseUp() {
43   document.getElementById("evento3").style.color = "orange";
44 }
45
46 function manejadorOnmouseover() {
47   document.getElementById("evento4").style.color = "yellow";
48 }
49
50 function manejadorOnmouseout() {
51   document.getElementById("evento4").style.color = "black";
```

```
52 }
53
54 function manejadorOnkeydown(evt) {
55     console.log("Manejador onkeydown\n");
56
57 }
58 function manejadorOnkeypress(evt) {
59
60     var charCode = event.keyCode || event.which;
61     var charStr = String.fromCharCode(charCode)
62     console.log("Manejador onkeypress: \n" + charStr);
63 }
64
65 function manejadorOnkeyup(evt) {
66    console.log("onkeyup handler");
67 }
68
69 function manejadorOntouchmove(event) {
70   var x,z;
71   var y,w;
72   x = event.touches[0].clientX;
73   y = event.touches[0].clientY;
74   document.getElementById("test").innerHTML = x + ", " + y;
75   if(event.touches.length>1){
76      z = event.touches[1].clientX;
77      w = event.touches[1].clientY;
78      document.getElementById("test2").innerHTML = z + ", " + w;
79   }
80 }
81
82 </script>
83
84 </body>
85 </html>
```

Código 24: Eventos en JavaScript

Salida:



1.10 APIs HTML5

El alcance de Javascript ha sido expandido con un grupo de poderosas librerías accesibles a través una serie de APIs, como por ejemplo:

1. **Canvas** Esta API es una API de dibujo, específica para la creación y manipulación de gráficos. Utiliza métodos Javascript predefinidos para operar.

2. **Drag and Drop** Esta API hace que arrastrar y soltar elementos con el ratón en la pantalla sea posible también en la web.

3. **Geolocation** Esta API tiene la intención de proveer acceso a información correspondiente con la ubicación física del dispositivo que está accediendo a la aplicación. Puede retornar datos como la latitud y longitud utilizando diferentes mecanismos

4. **Web Storage** Esta API introduce dos atributos para almacenar datos en el ordenador del usuario: sessionStorage y localStorage. El atributo sessionStorage permite a los desarrolladores hacer un seguimiento de la actividad de los usuarios almacenando información que estará disponible en cada instancia de la aplicación durante la duración de la sesión. El atributo localStorage, por otro lado, ofrece a los desarrolladores un área de almacenamiento, creada para cada aplicación, que puede conservar varios megabytes de información, preservando de este modo información y datos en el ordenador del usuario de forma persistente.

5. **File** Este es un grupo de APIs desarrollada para proveer la capacidad de leer, escribir y procesar archivos de usuario.

6. **XMLHttpRequest Level 2** Esta API es una mejora de la vieja XMLHttpRequest destinada a la construcción de aplicaciones Ajax. Incluye nuevos métodos para controlar el progreso de la operación y realizar solicitudes cruzadas (desde diferentes orígenes).

7. **WebSockets** Esta API provee un mecanismo de comunicación de dos vías entre clientes y servidores para generar aplicaciones en tiempo real como salas de chat o juegos en línea.

8. **Web Workers** Esta API potencia Javascript permitiendo el procesamiento de código detrás de escena, de forma separada del código principal, sin interrumpir la actividad normal de la página web, incorporando la capacidad de multitarea a este lenguaje.

9. **Offline** Esta API apunta a mantener las aplicaciones funcionales incluso cuando el dispositivo es desconectado de la red.

1.11 Canvas

El elemento <**canvas**> es utilizado para dibujar gráficos en una página web mediante JavaScript. Canvas tiene varios métodos para dibujar trayectorias, rectángulos, círculos, texto, imágenes, etc. El elemento fue originalmente introducido por Apple en el OS X Dashboard y Safari. Internet Explorer, antes de la versión 9.0 beta, no admite de forma nativa <**canvas**>.

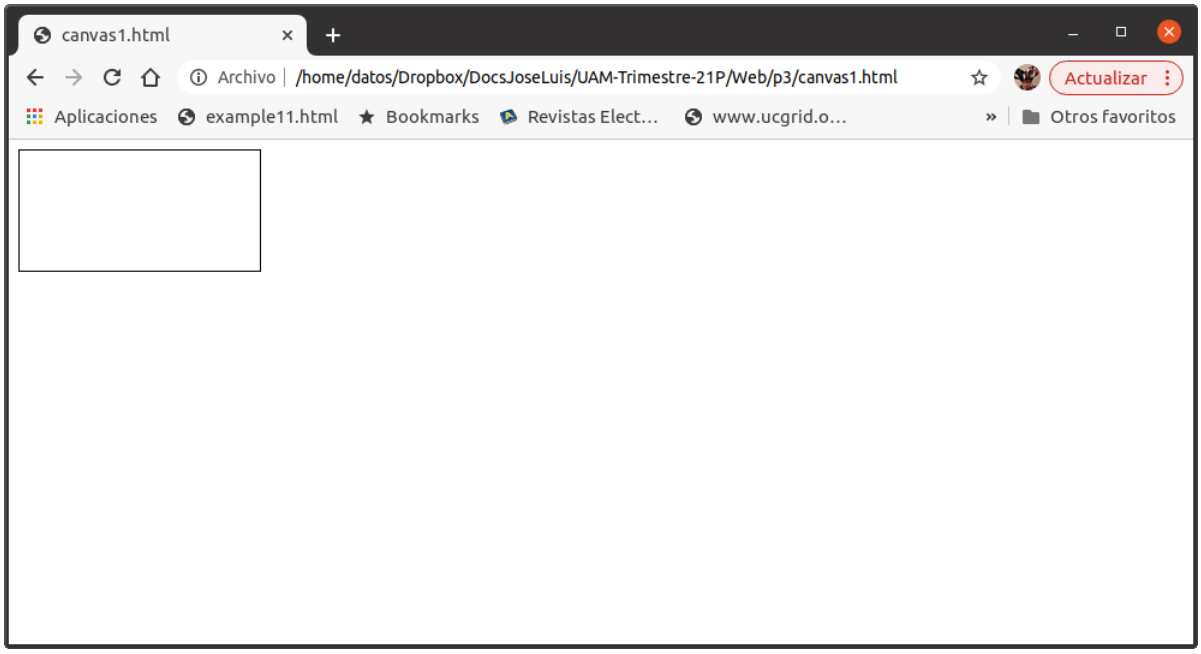
1.11.1 El área del canvas

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;"
    >
6 Tu navegador no soporta canvas
7 </canvas>
```

```
8
9  </body>
10 </html>
```

Código 25: Área del canvas

Salida:

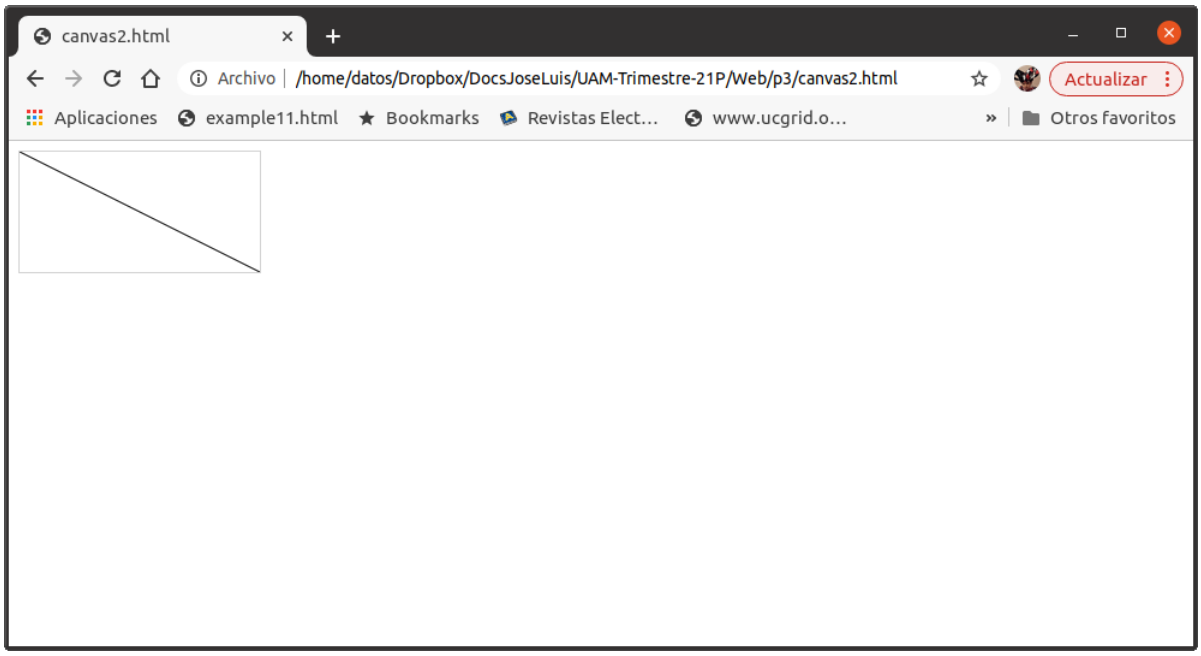


1.11.2 Una línea en canvas

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <canvas id="canvas" width="200" height="100" style="border:1px solid #d3d3d3;">
6  Tu navegador no soporta canvas.</canvas>
7
8  <script>
9  var c = document.getElementById("canvas");
10 var ctx = c.getContext("2d");
11 ctx.moveTo(0,0);
12 ctx.lineTo(200,100);
13 ctx.stroke();
14 </script>
15
16 </body>
17 </html>
```

Código 26: Línea en canvas

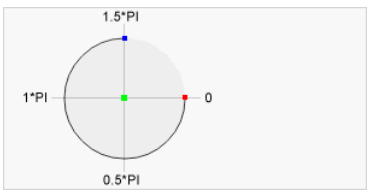
Salida:



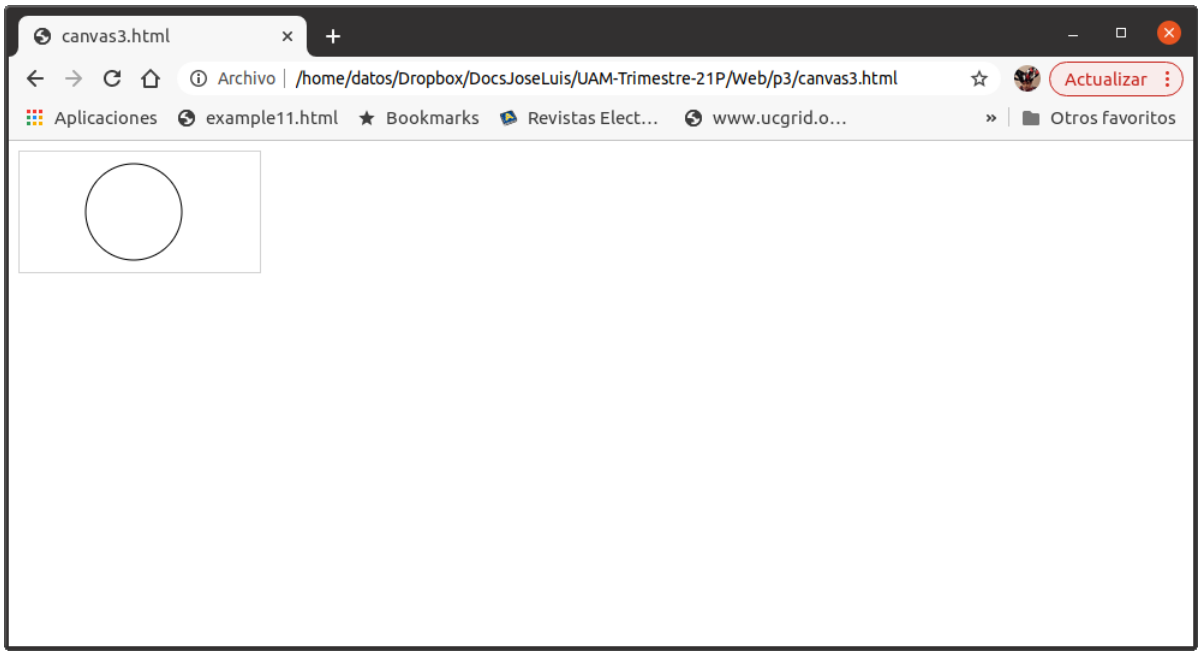
1.11.3 Un círculo en canvas

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <canvas id="canvas" width="200" height="100" style="border:1px solid #d3d3d3;">
6  El navegador no soporta canvas </canvas>
7
8  <script>
9  var c = document.getElementById("canvas");
10 var ctx = c.getContext("2d");
11 ctx.beginPath();
12 ctx.arc(95,50,40,0,2*Math.PI);
13 ctx.stroke();
14 </script>
15
16 </body>
17 </html>
```

Código 27: Circulo en canvas



Salida:



1.11.4 Una animación

Descargar fuente.

1.12 Drag and drop

El API Drag and Drop, introducida por la especificación HTML5, permite arrastrar un elemento desde un lugar y luego soltarlo en otro.

1.12.1 Eventos Drag&Drop

Uno de los más importantes aspectos de esta API es un conjunto de siete nuevos eventos introducidos para informar sobre cada una de las situaciones involucradas en el proceso. Algunos de estos eventos son disparados por la fuente (el elemento que es arrastrado) y otros son disparados por el destino (el elemento en el cual el elemento arrastrado será soltado).

1. **dragstart**: Este evento es disparado en el momento en el que el arrastre comienza.
2. **drag**: Este evento es similar al evento mousemove, excepto que será disparado durante una operación de arrastre por el elemento origen.
3. **dragend**: Cuando la operación de arrastrar y soltar finaliza (sea la operación exitosa o no) este evento es disparado por el elemento origen.
4. **dragenter**: Este evento es disparado, cuando el puntero del ratón entra dentro del área ocupada por los posibles elementos destino
5. **dragover** Este evento es similar al evento mousemove, excepto que es disparado durante una operación de arrastre por posibles elementos destino.

6. **drop**: Cuando el elemento origen es soltado durante una operación de arrastrar y soltar, este evento es disparado por el elemento destino.

7. **dragleave**: Este evento es disparado cuando el ratón sale del área ocupada por un elemento durante una operación de arrastrar y soltar.

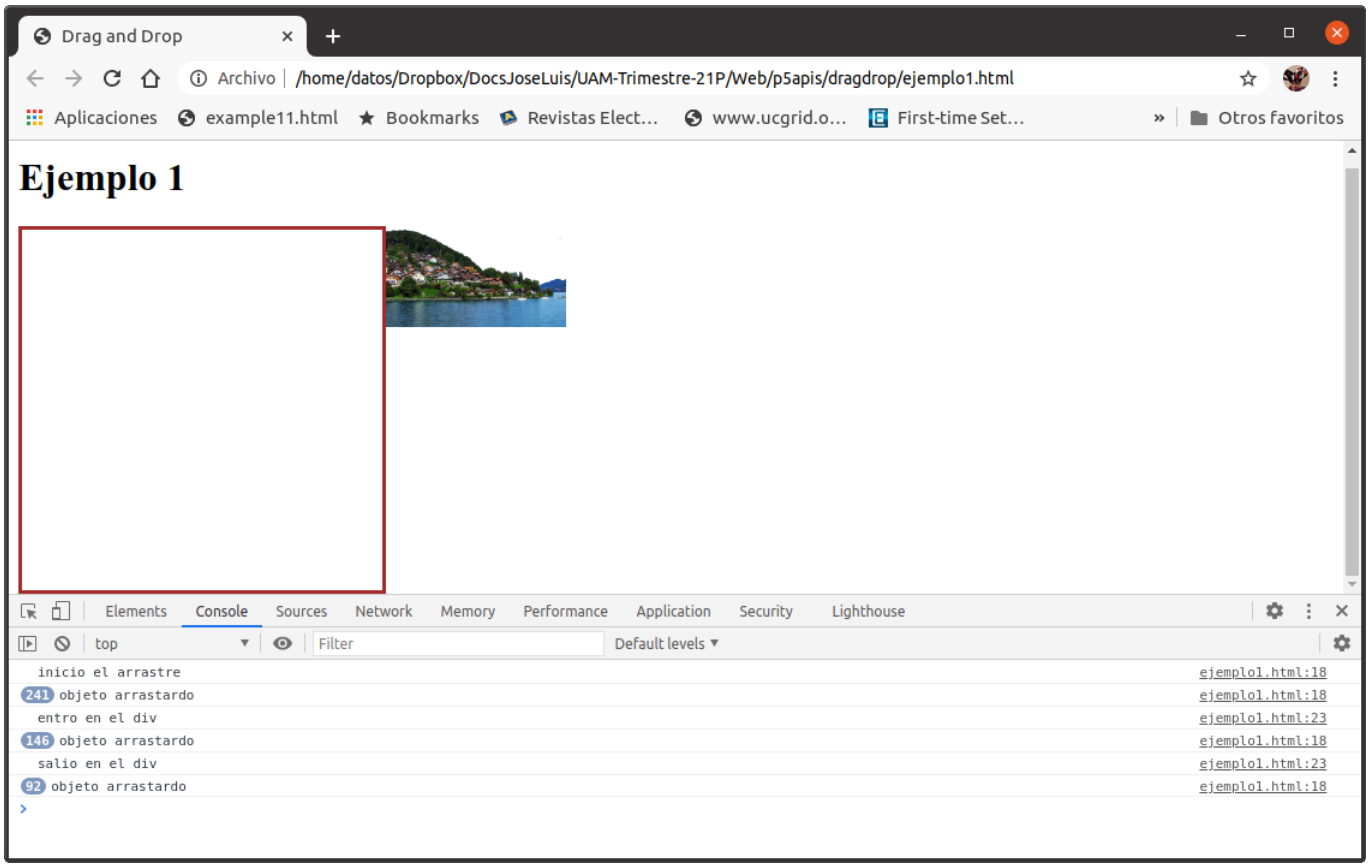
IMPORTANTE: existe un aspecto importante que debemos considerar. Los navegadores realizan acciones por defecto durante una operación de arrastrar y soltar. Para algunos eventos, como dragenter, dragover y drop, la prevención es necesaria, incluso cuando una acción personalizada ya fue especificada. Para eliminar un comportamiento por default se puede hacer uso del método del objeto event preventDefault();

1.12.2 Eventos

```
<!DOCTYPE HTML>
<html>
<head>
<title>Drag and Drop</title>

<style type="text/css">

   #contenedor { border: solid brown; width:300px; height:300px; float:left;}

</style>
</head>

<body>
<h1>Ejemplo 1</h1>

<img src="img.png" width="150" height:'150' draggable='true'
    ondrag="console.log('objeto arrastrado')"
    ondragstart="console.log('inició el arrastre')">

<div id='contenedor'
     ondragenter="console.log('entró en el div')"
     ondragleave="console.log('salió en el div')"
</div>

</body>

</html>
```

Código 28: Eventos drag and drop

Salida:



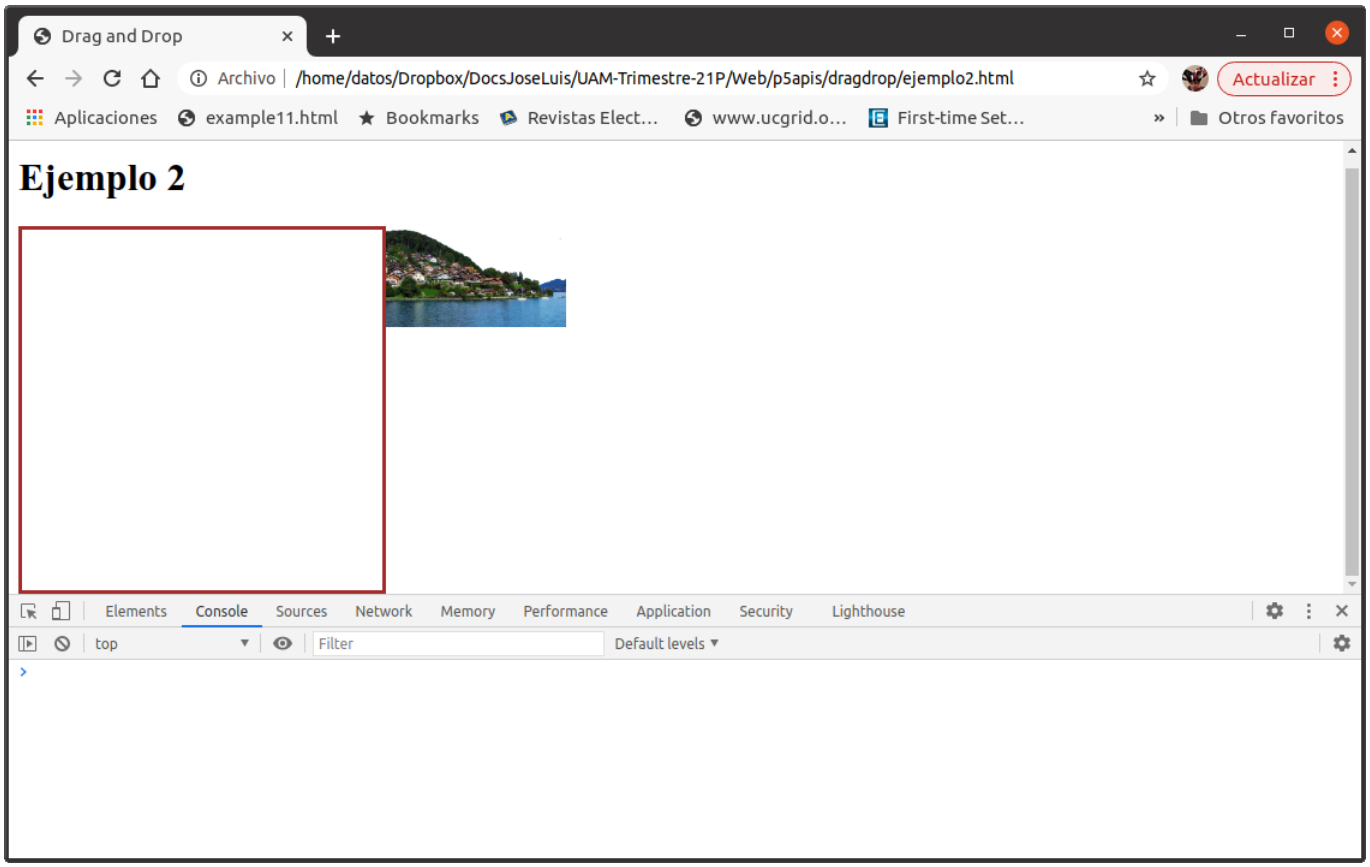
1.12.3 Arrastrar y soltar

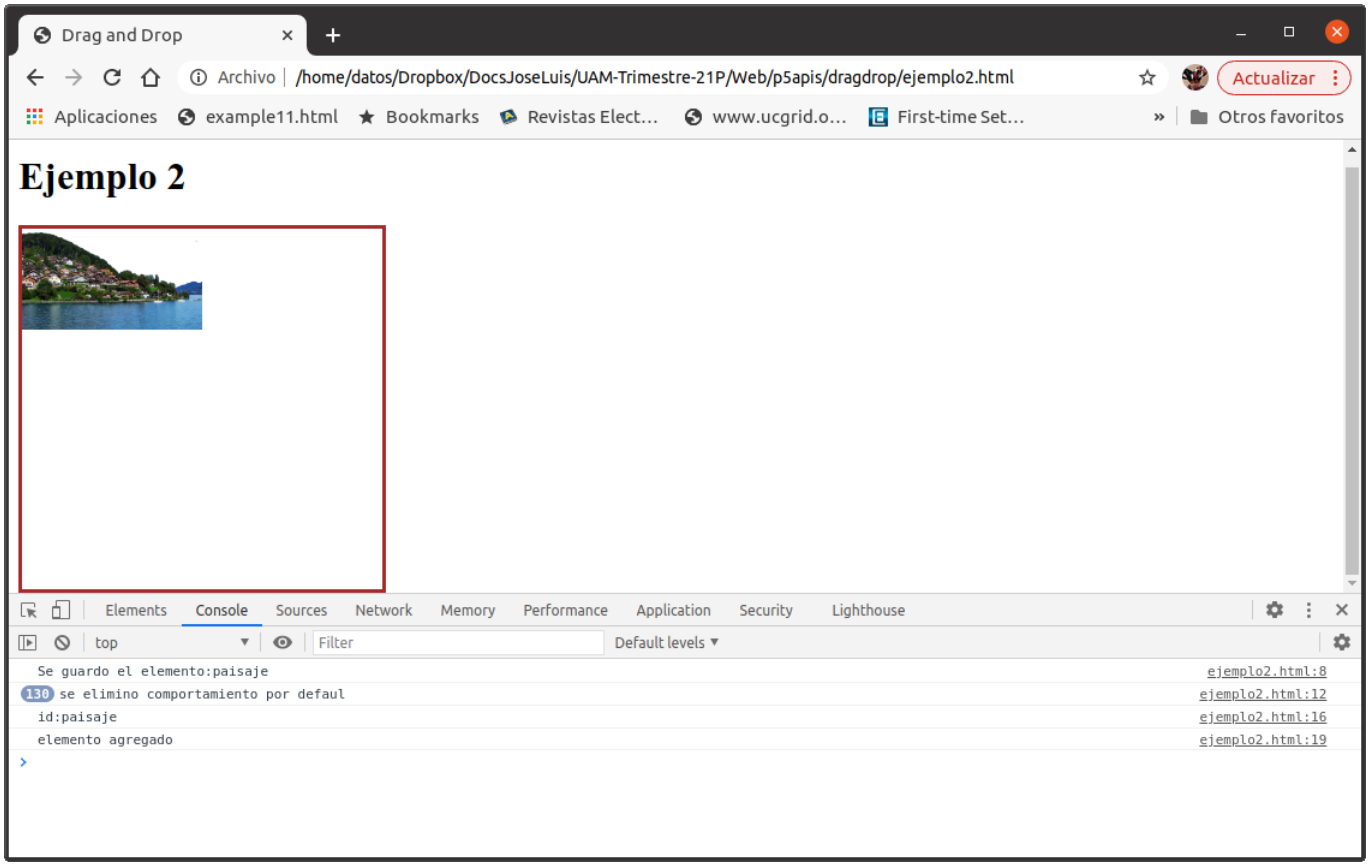
```
<!DOCTYPE HTML>
<html>
<head>
<title>Drag and Drop</title>
<script type="text/javascript">
function iniciaArrastre(e){
    e.dataTransfer.setData("villa", e.target.id);
    console.log("Se guardo el elemento:"+e.target.id);
}
function entrando(e){
    e.preventDefault();
    console.log("se elimino comportamiento por defaul");
}
function soltado(e){
    var id = e.dataTransfer.getData("villa");
    console.log("id:"+id);
    var objeto=document.getElementById(id);
    e.target.appendChild(objeto);
    console.log("elemento agregado");
}

</script>
<style type="text/css">
```

```
24
25      #contenedor {
26          border: solid brown;
27          width:300px;
28          height:300px;
29          float:left;
30      }
31
32  </style>
33  </head>
34
35  <body>
36  <h1>Ejemplo 2</h1>
37
38  <img id="paisaje" src="img.png" width="150" height:'150' draggable='true'
        ondragstart="iniciaArrastre(event)">
39
40  <div id='contenedor'
41       ondragover="entrando(event)"
42       ondrop="soltado(event)"
43  </div>
44
45  </body>
46
47  </html>
```

Código 29: Arrastrar y soltar elemento.

Salida:





1.12.4 Varios elementos

```html
<!DOCTYPE html>
<html>
<head>
   <title>Drag and Drop</title>
   <script src="script.js"> </script>
   <link rel="stylesheet" href="estilo.css">
</head>
<body>
      <section id="cajasoltar"> Arrastre y suelte las imágenes aquí  </section>
      <section id="cajaimagenes">
      <img id="imagen1" src="paisaje1.png">
      <img id="imagen2" src="paisaje2.png">
      <img id="imagen3" src="paisaje3.png">
      <img id="imagen4" src="paisaje4.png">
      </section>
</body>
</html>
```

Código 30: Arrastrar y soltar varios elementos

```javascript
function iniciar(){

   var imagenes=document.querySelectorAll('#cajaimagenes > img');
   for(var i=0; i<imagenes.length; i++){
      imagenes[i].addEventListener('dragstart', arrastrado, false);
   }
   soltar=document.getElementById('cajasoltar');
   soltar.addEventListener('dragover', function(e){
      e.preventDefault();
      }, false
    );
    soltar.addEventListener('drop', soltado, false);
}

function arrastrado(e){
   elemento=e.target;
   e.dataTransfer.setData('Text', elemento.getAttribute('id'));
}

function soltado(e){
   e.preventDefault();
   var src=null;
   var id=e.dataTransfer.getData('Text');
   if(id!="imagen4"){
      src=document.getElementById(id).src;
      soltar.innerHTML='<img src="'+src+'">';
    }else{
```

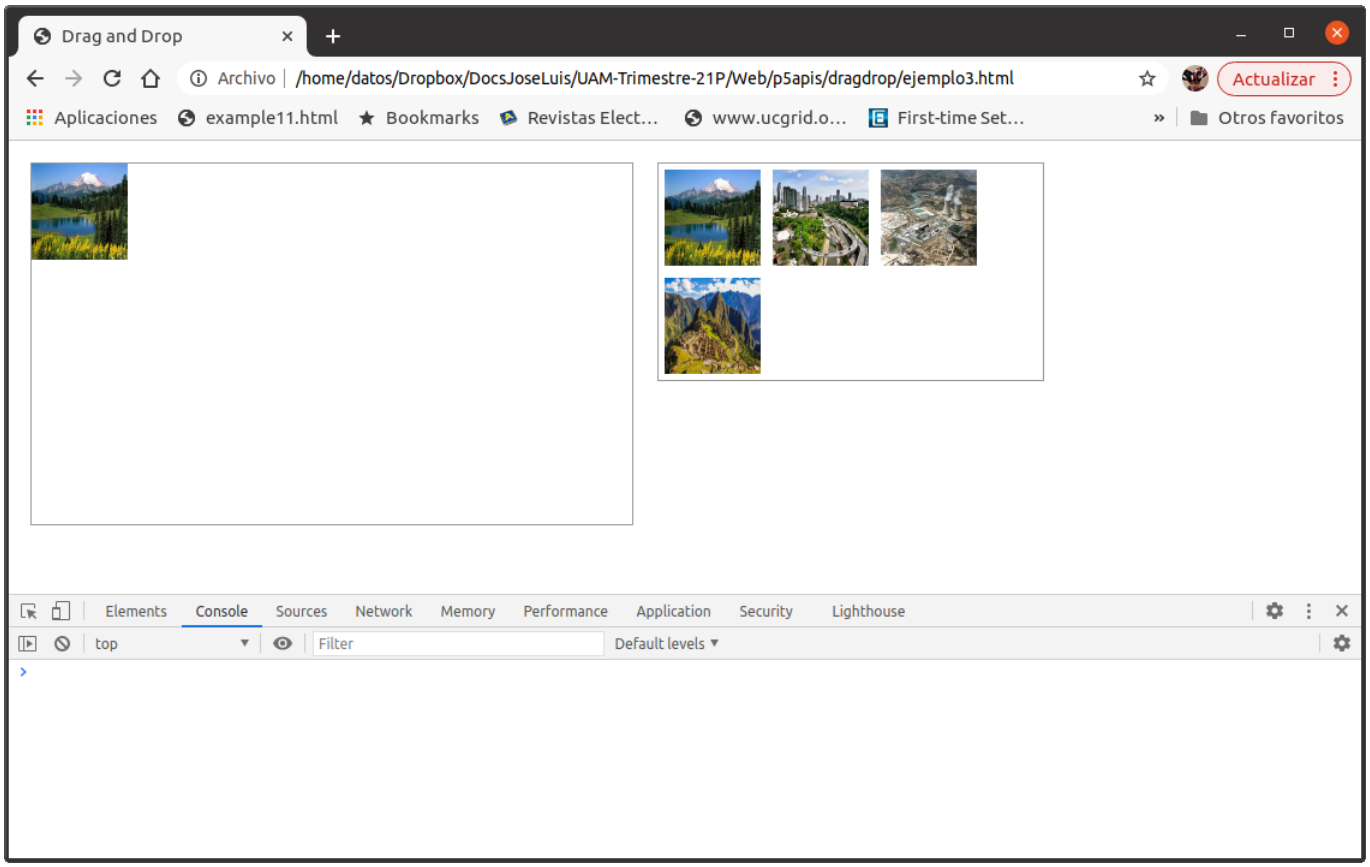
```
28        soltar.innerHTML='la imagen no es admitida';
29      }
30  }
31  window.addEventListener('load', iniciar, false);
```

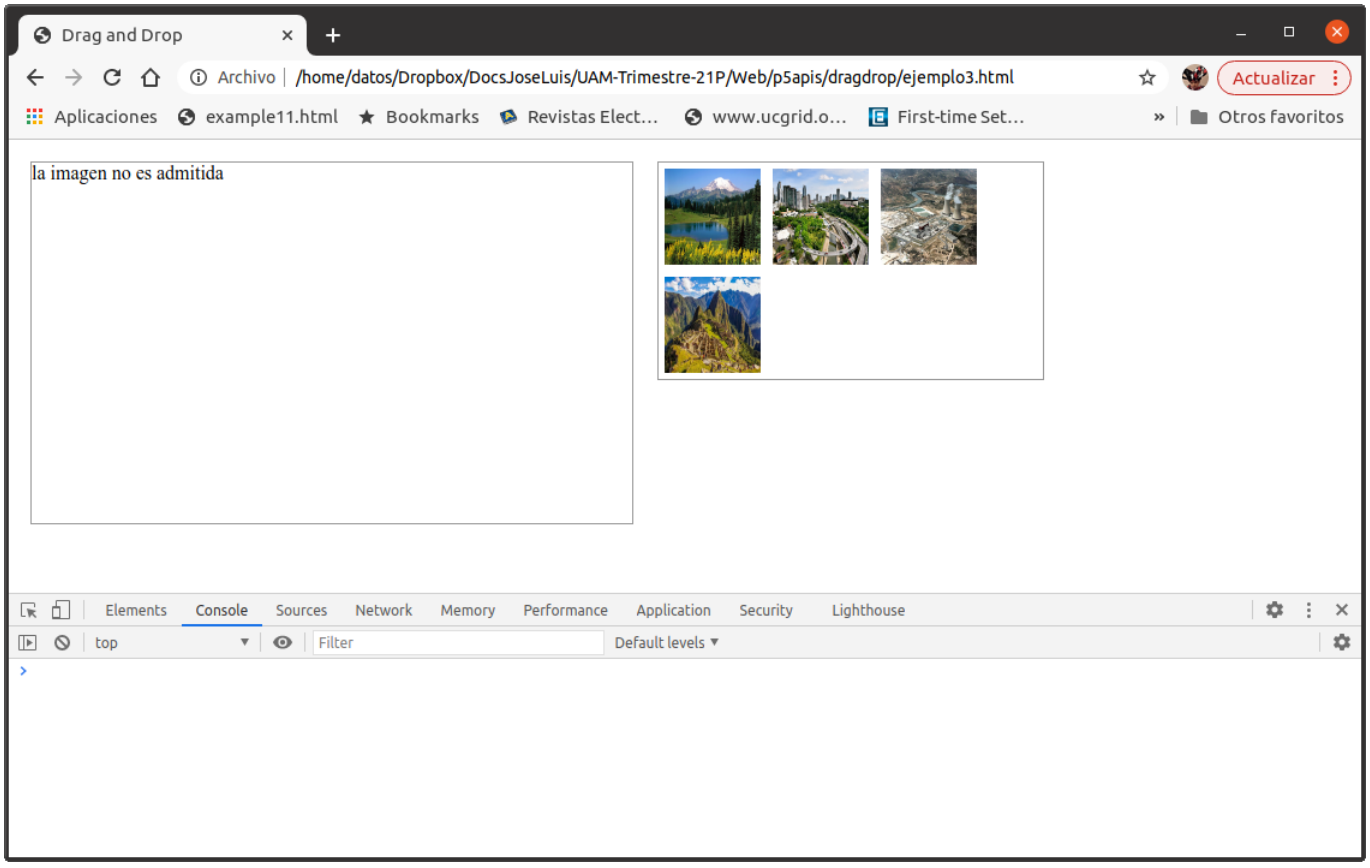
Código 31: Arrastrar y soltar varios elementos (JavaScript).

```
1   #cajasoltar{
2      float: left;
3      width: 500px;
4      height: 300px;
5      margin: 10px;
6      border: 1px solid #999999;
7   }
8   #cajaimagenes{
9      float: left;
10     width: 320px;
11     margin: 10px;
12     border: 1px solid #999999;
13  }
14  #cajaimagenes > img{
15     float: left;
16     padding: 5px;
17  }
18  img {
19     width: 80px;
20     height: 80px;
21  }
```

Código 32: Arrastrar y soltar varios elementos (CSS)

Salida:


Drag and Drop
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p5apis/dragdrop/ejemplo3.html
Actualizar
Aplicaciones
example11.html
Bookmarks
Revistas Elect...
www.ucgrid.o...
First-time Set...
Otros favoritos
Elements
Console
Sources
Network
Memory
Performance
Application
Security
Lighthouse
top
Filter
Default levels


Drag and Drop
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p5apis/dragdrop/ejemplo3.html
Actualizar
Aplicaciones
example11.html
Bookmarks
Revistas Elect...
www.ucgrid.o...
First-time Set...
Otros favoritos
la imagen no es admitida
Elements
Console
Sources
Network
Memory
Performance
Application
Security
Lighthouse
top
Filter
Default levels

1.12.5 Ejercicio

Crea un documento HTML en la que se tengan tres imágenes: dos borregos y un lobo. Si hay al menos un borrego el lobo no pueda entrar al contenedor pero el otro borrego si. En caso de que sea el lobo el primero en entrar entonces los borregos ya no pueden entrar.

1.13 Web Storage

WebStorage nos ofrece una alternativa a el uso de las cookies, que a diferencia de estas, el límite de almacenamiento es mucho más grande (por lo menos 5 MB) y la información nunca se transfiere al servidor.

WebSatorage define dos objetos principales para el almacenamiento de datos:

1. **window.sessionStorage:**almacena los datos para una sola sesión (los datos se pierden cuando la pestaña del navegador se cierra).
2. **window.localStorage:** almacena datos sin fecha de vencimiento.

1.13.1 Session Storage

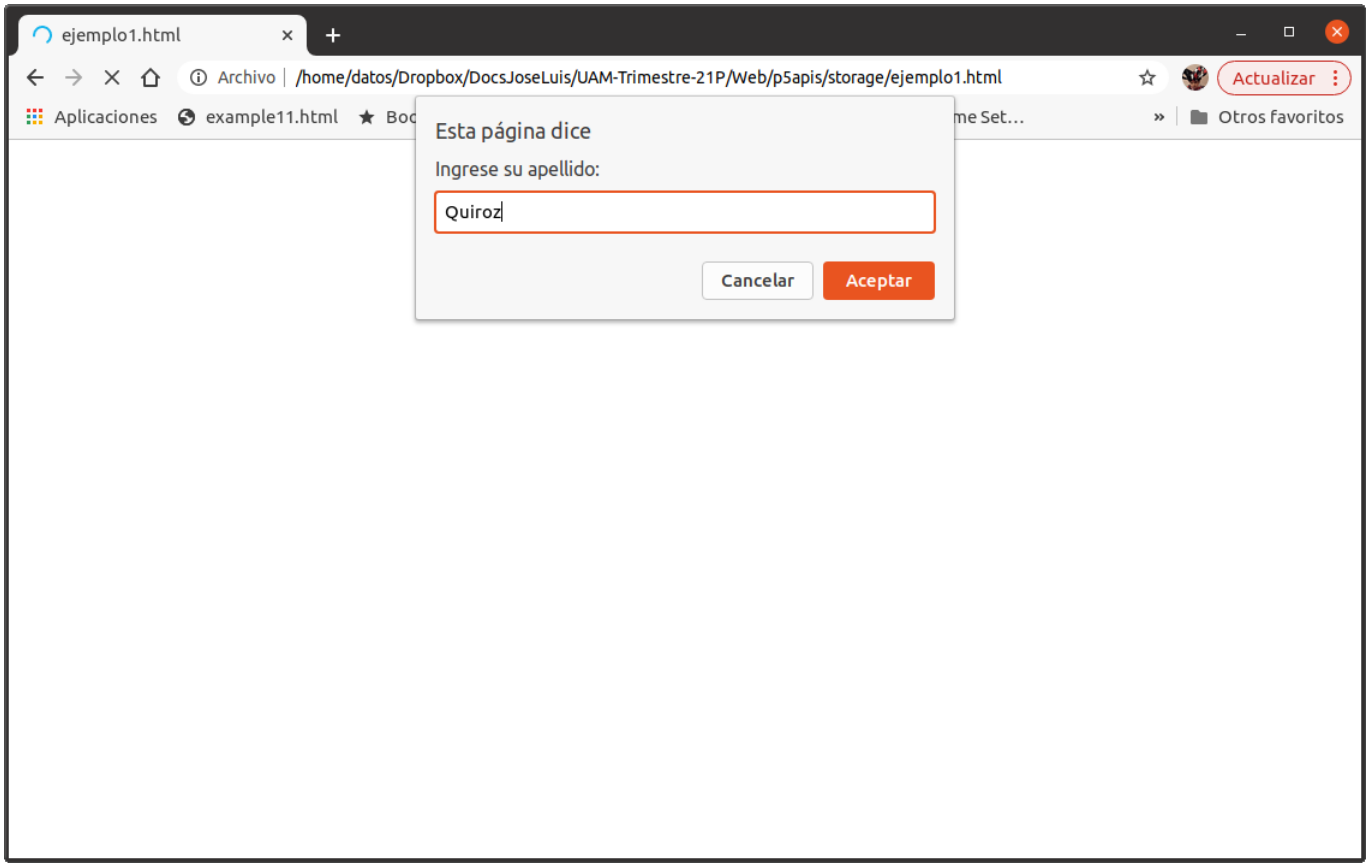
El siguiente ejemplo utiliza el objeto sessionStorage, prueba el código recargando la página varias veces. Observa como solo la primera vez que se carga la página se solicita el apellido pero al volver a recargar la página dicho dato ya no es solicitado, solo cuando se cierra la pestaña y se vuelve a abrir la página se pide de nuevo ya que dicho valor se pierde.

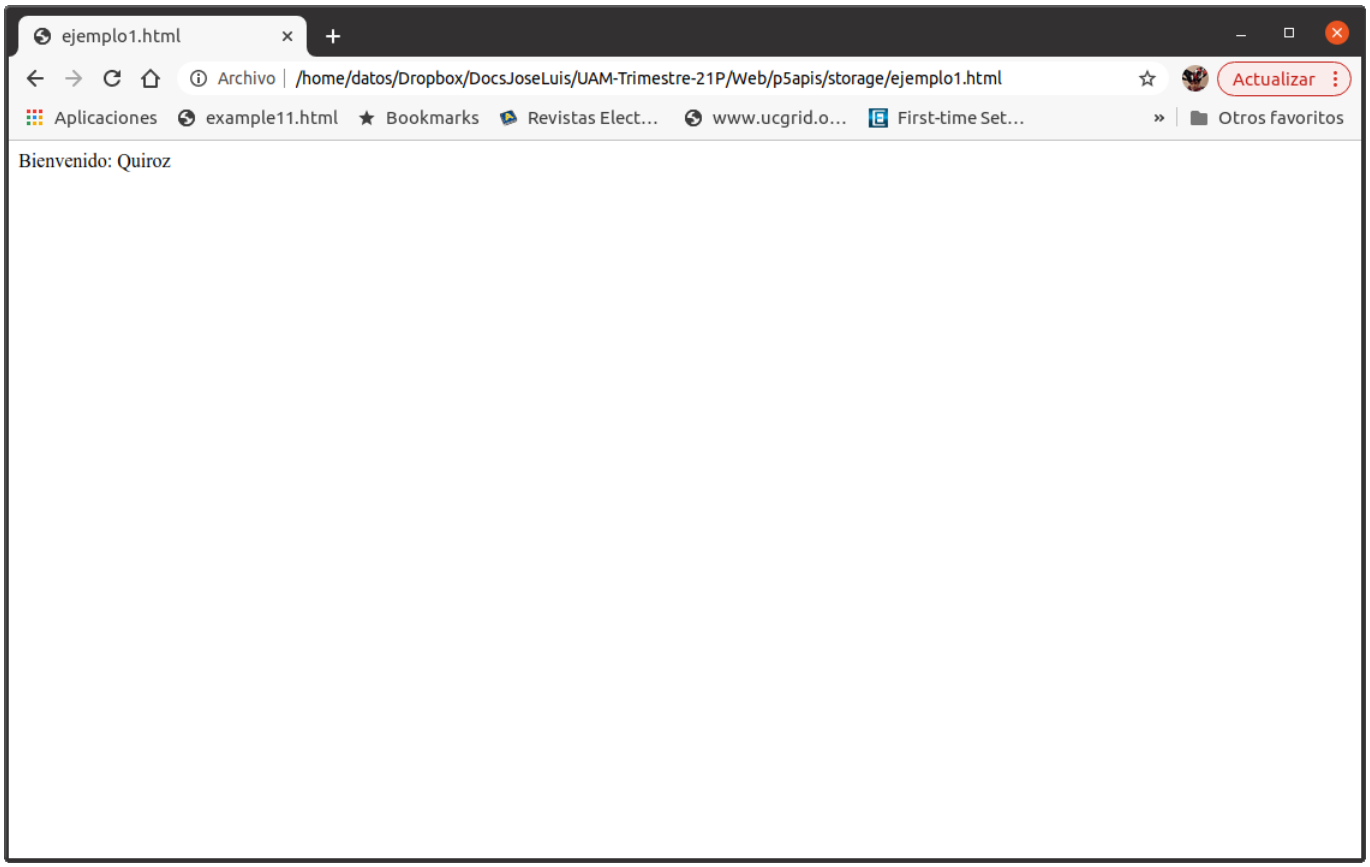
```
1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <div id="result"></div>
6
7  <script>
8  /* Verifica soporte*/
9  function almacenaDatos(){
10
11    if(typeof(Storage) !== undefined) {
12    /* Almacenamiento Local*/
13       if( sessionStorage.getItem("apellido")==undefined){
14          apellido=prompt('Ingrese su apellido:','');
15          sessionStorage.setItem("apellido",apellido);
16          document.getElementById("result").innerHTML = "Bienvenido: "+
                sessionStorage.getItem("apellido");
17       }else{
18          console.log("apellido= "+sessionStorage.getItem("apellido"));
19          alert("El nombre fue:"+sessionStorage.getItem("apellido"));
```

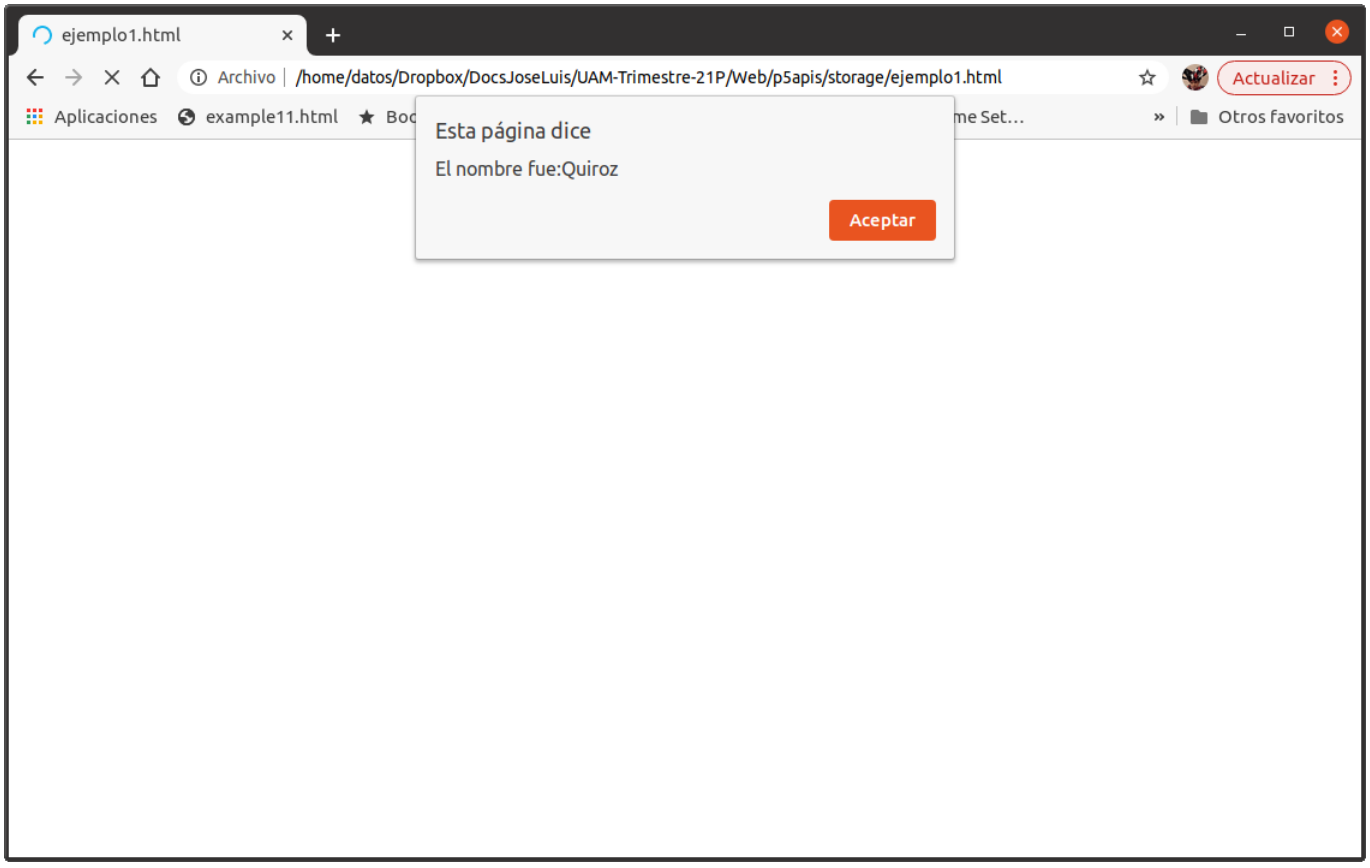
```
20        }
21
22     }else {
23        document.getElementById("result").innerHTML = "Lo siento no se puede
              almacenar el nombre :(";
24     }
25  }
26  window.onload=almacenaDatos;
27  </script>
28  </body>
29  </html>
```

Código 33: Ejemplo de un almacenamiento por sesión

Salida:




ejemplo1.html
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p5apis/storage/ejemplo1.html
Actualizar
Aplicaciones
example11.html
Bookmarks
Revistas Elect...
www.ucgrid.o...
First-time Set...
Otros favoritos
Bienvenido: Quiroz


ejemplo1.html
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p5apis/storage/ejemplo1.html
Actualizar
Aplicaciones
example11.html
Otros favoritos
Esta página dice
El nombre fue:Quiroz
Aceptar

1.13.2 Local Storage

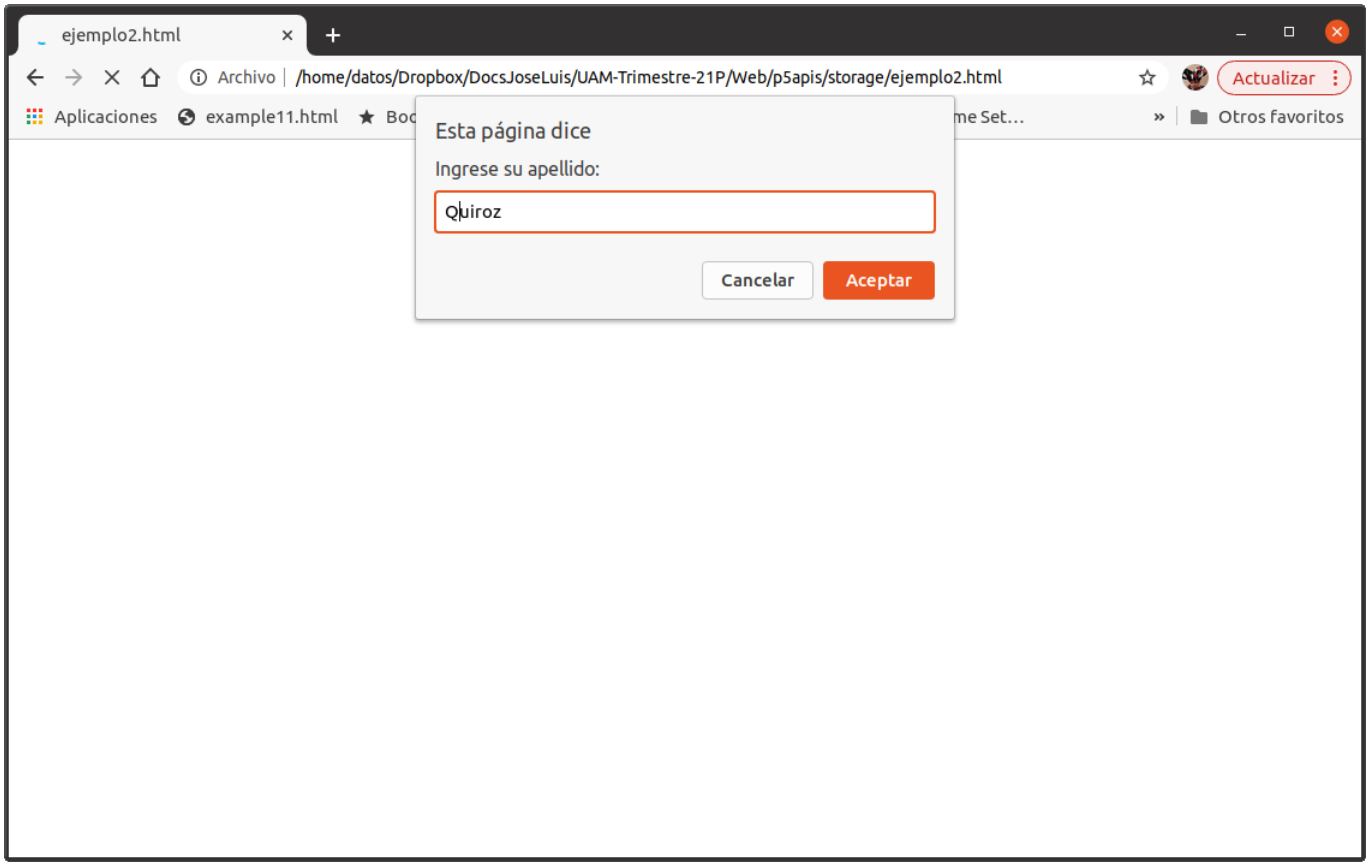
En este ejemplo se extiende el ejemplo anterior de tal forma que se contabiliza por medio de localStorage las veces que se ha visitado la página con el mismo apellido. Se incorpora un botón que elimina el alacenamiento para que se reinicie el conteo.

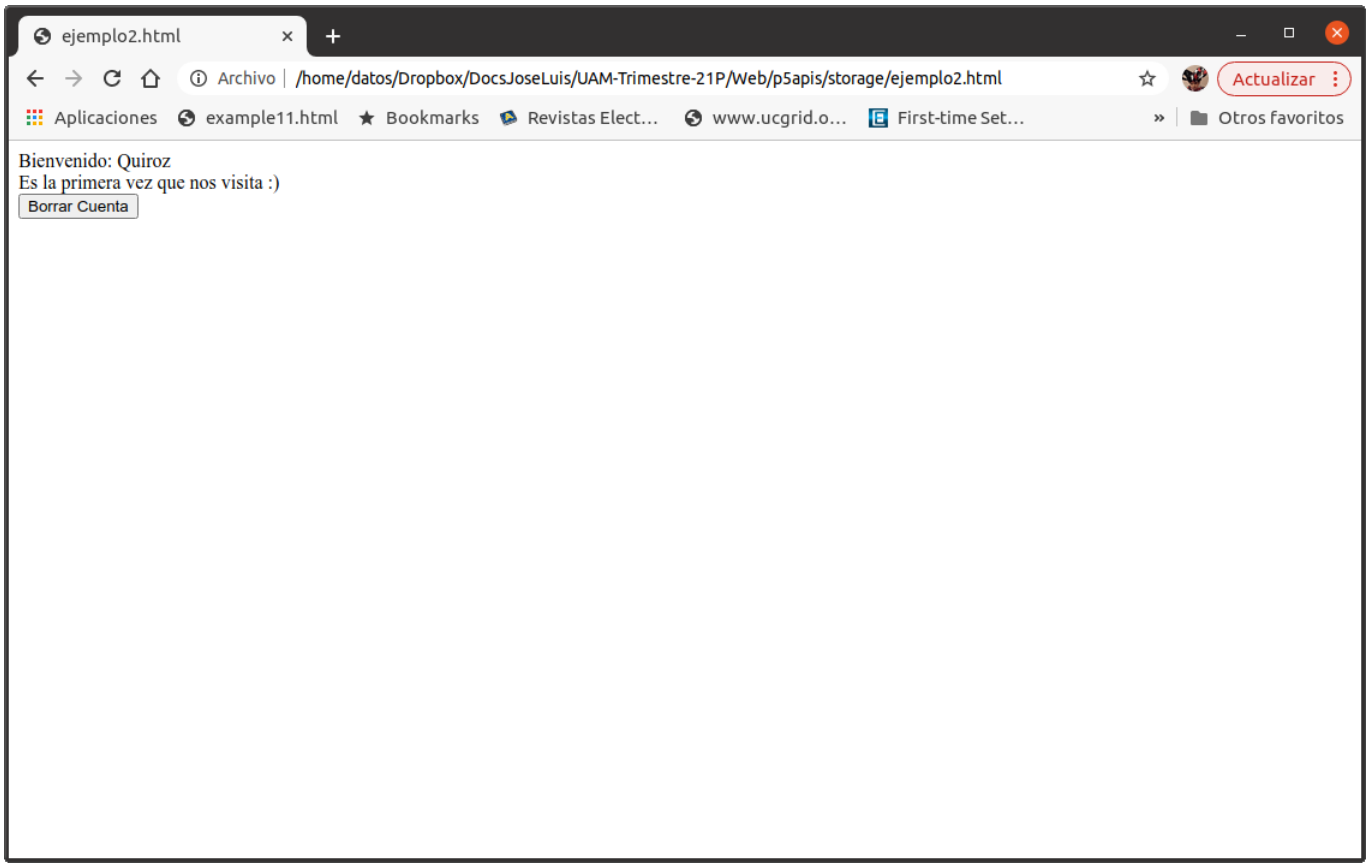
```
<!DOCTYPE html>
<html>
<body>

<div id="result"></div>

<script>
function almacenaDatos(){

   if (typeof(Storage) !== undefined) {

      if( sessionStorage.getItem("apellido")==undefined){
         apellido=prompt('Ingrese su apellido:','');
         sessionStorage.setItem("apellido",apellido);
      }else{
         apellido=sessionStorage.getItem("apellido");
         console.log("apellido= "+sessionStorage.getItem("apellido"));
      }
      document.getElementById("result").innerHTML = "Bienvenido: "+
          sessionStorage.getItem("apellido");
      if( localStorage.getItem(apellido)==undefined){
         console.log("no existe");
         localStorage.setItem(apellido,'0');
         texto=document.getElementById("result").innerHTML;
         texto+="<br>Es la primera vez que nos visita :)";
      }else{
         console.log("existe");
         localStorage.setItem(apellido,Number(localStorage.getItem(apellido))+1)
             ;
         texto=document.getElementById("result").innerHTML;
         texto+="<br>Nos has visitado "+localStorage.getItem(apellido)+" veces";
      }
      document.getElementById("result").innerHTML=texto;

   } else {
      document.getElementById("result").innerHTML = "Lo siento no se puede
          almacenar datos en el navegador :(";
   }
}

function borrar(){
```

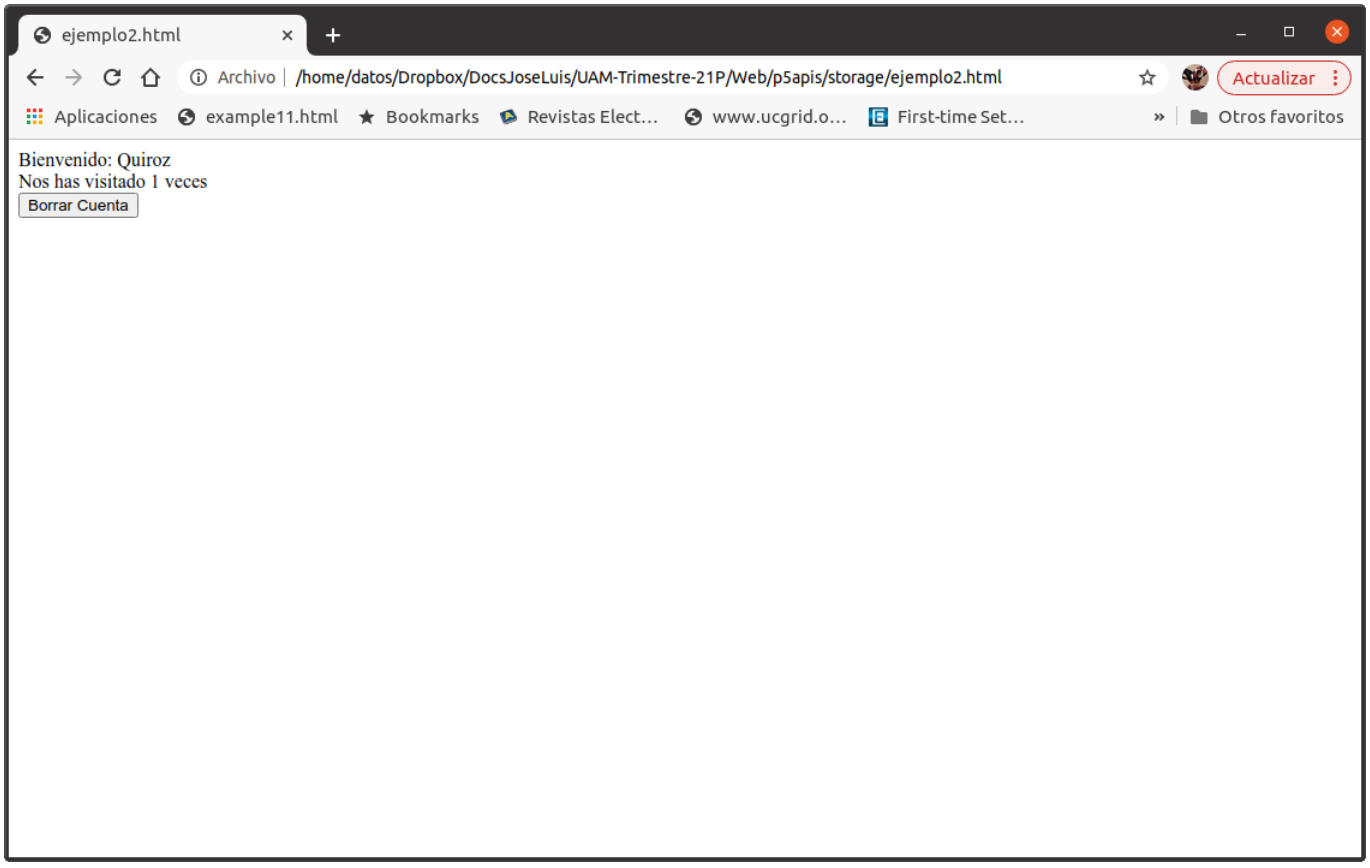
```
39
40      if( localStorage.getItem(apellido)!==undefined){
41
42           localStorage.removeItem(apellido);
43           alert("Limpiando local storage");
44       }
45
46  }
47
48  window.onload=almacenaDatos;
49  </script>
50
51
52  <div id="result"></div>
53
54  <input type=button onclick=borrar() value="Borrar Cuenta">
55
56
57  </body>
58  </html>
```

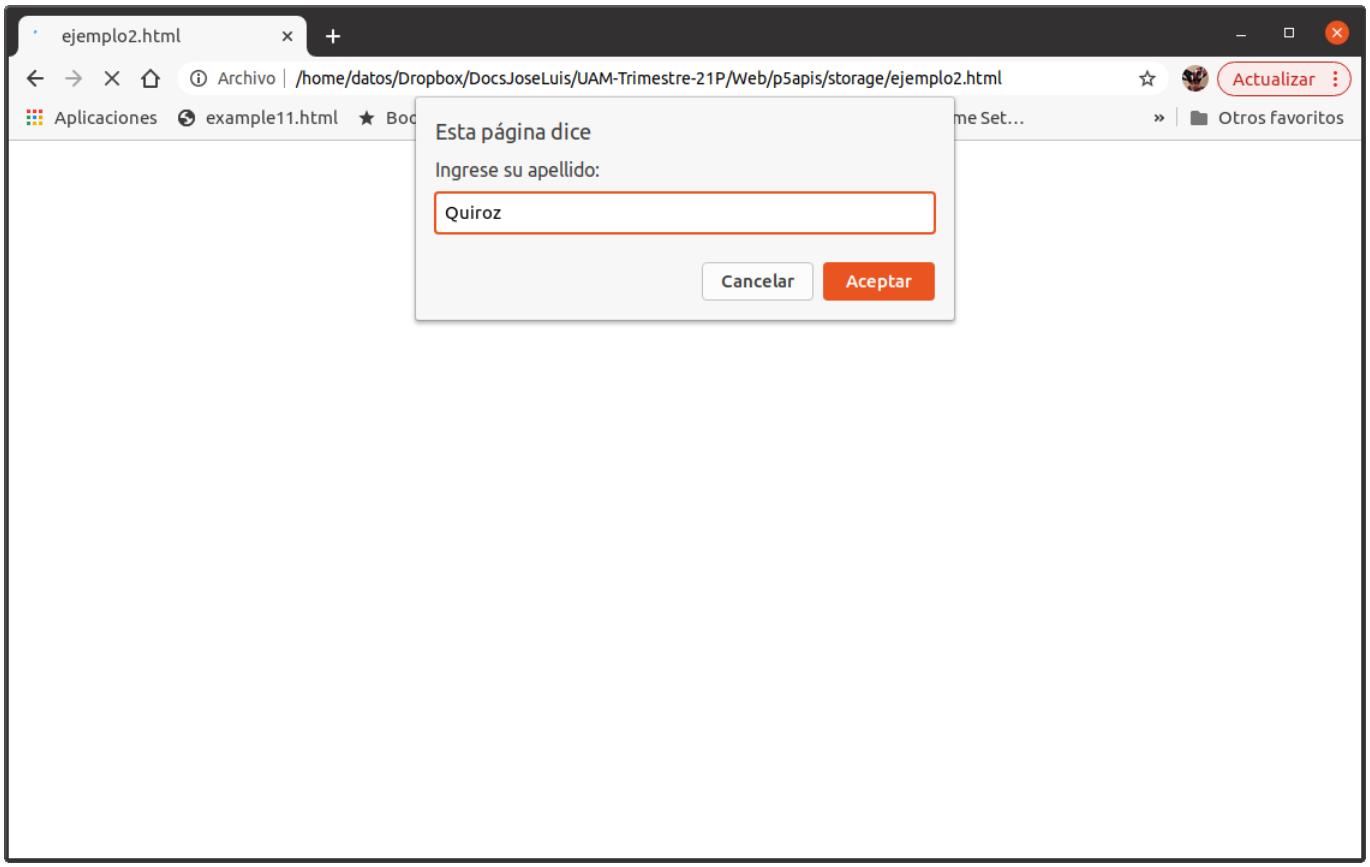
Código 34: Ejemplo de un almacenamiento local

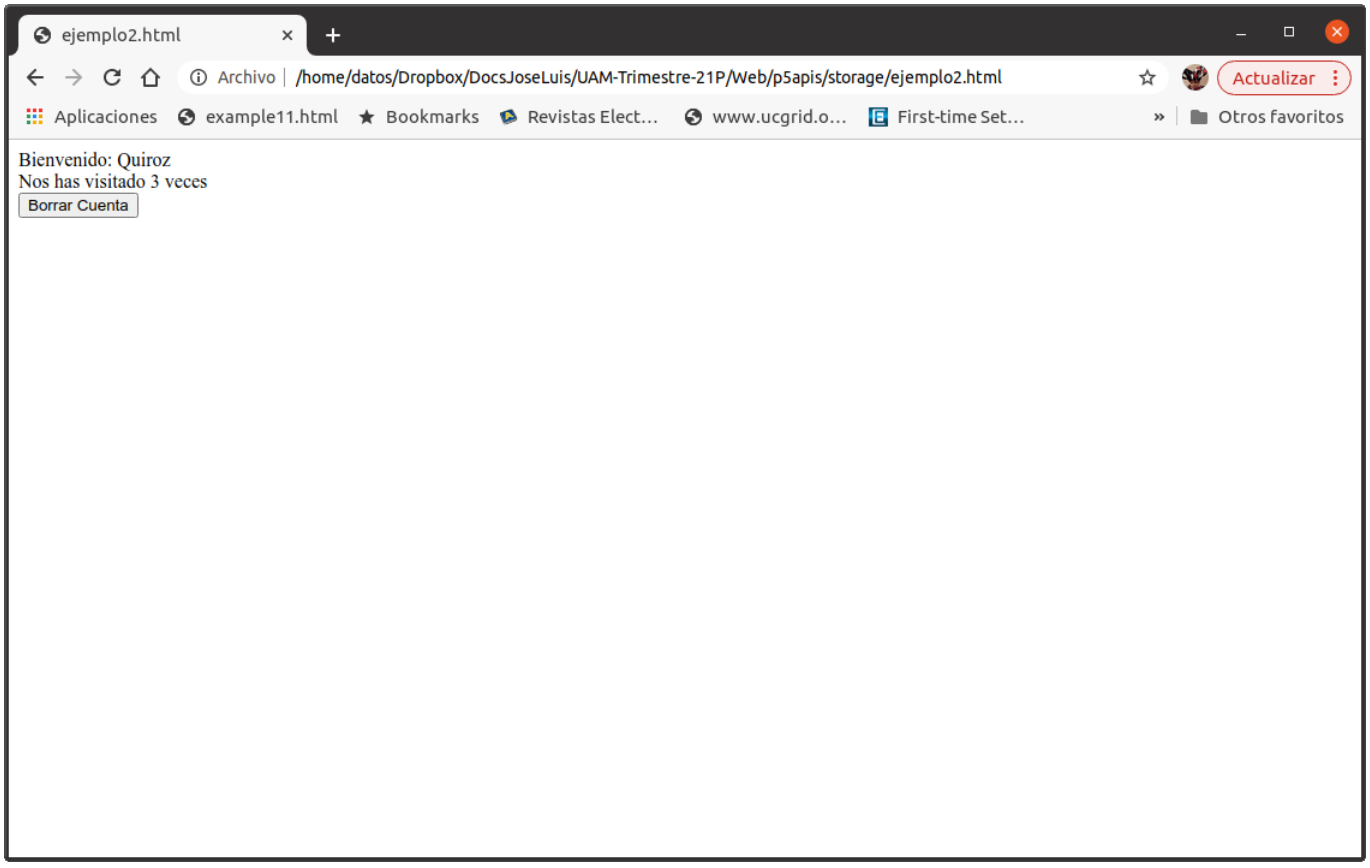
Salida:




ejemplo2.html
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p5apis/storage/ejemplo2.html
Actualizar
Aplicaciones
example11.html
Bookmarks
Revistas Elect...
www.ucgrid.o...
First-time Set...
Otros favoritos
Bienvenido: Quiroz
Es la primera vez que nos visita :)
Borrar Cuenta


ejemplo2.html
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p5apis/storage/ejemplo2.html
Actualizar
Aplicaciones
example11.html
Bookmarks
Revistas Elect...
www.ucgrid.o...
First-time Set...
Otros favoritos
Bienvenido: Quiroz
Nos has visitado 1 veces
Borrar Cuenta


ejemplo2.html
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p5apis/storage/ejemplo2.html
Actualizar
Aplicaciones
example11.html
Otros favoritos
Esta página dice
Ingrese su apellido:
Quiroz
Cancelar
Aceptar


ejemplo2.html
Archivo | /home/datos/Dropbox/DocsJoseLuis/UAM-Trimestre-21P/Web/p5apis/storage/ejemplo2.html
Actualizar
Aplicaciones
example11.html
Bookmarks
Revistas Elect...
www.ucgrid.o...
First-time Set...
Otros favoritos
Bienvenido: Quiroz
Nos has visitado 3 veces
Borrar Cuenta

1.13.3 Ejercicio

Revisar el código del documento de burbuja y agregar un **score** cada que se explota un burbuja. El **score** se almacena en **localStorage** de tal forma que no se pierde el puntaje.